НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра автоматики та управління в технічних системах
(повна назва кафедри)

«На правах рукопису»
УДК _____

«До захисту допущено»

Завідувач кафедри

_____ _____
(підпис) (ініціали, прізвище)

“___”_____20__ р.

Магістерська дисертація

зі спеціальності (спеціалізації)126 Інформаційні системи та технології

на тему: Система збору та аналізу тексових даних з соціальних мереж_____
_____

Виконав: студент __6__ курсу, групи ___ІА-з82мп_____
(шифр групи)

Погорілий Богдан Анатолійович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник: завідувач кафедри, д.т.н., професор Ролік О. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант: _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент:_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) Інформатики та обчислювальної техніки
(повна назва)

Кафедра Автоматики та управління в технічних системах
(повна назва)

Освітньо-кваліфікаційний ступінь другий (магістерський)
(назва ОКР)

Спеціальність 126 Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедри
О.І. Ролік
(підпис) (ініціали, прізвище)
« » 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Погорілому Богдану Анатолійовичу

1. Тема дисертації Система збору та аналізу текстових даних з соціальних мереж

Науковий керівник дисертації Ролік О. І. д.т.н., проф.
затверджені наказом по університету від «____» _____ 2019 р. № _____

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження інтелектуальні алгоритми аналізу текстових коментарів

4. Зміст пояснювальної записки а) аналіз існуючих методів оброблення текстових даних; б) вибір методів оброблення натуральних мов; в) вибір технічних засобів та реалізація системи; г) тестування та особливості роботи системи д) розробка стартап проекту.

5. Перелік завдань, які потрібно розробити: аналізування предметної області, аналізування існуючих рішень, дослідження та аналізування методів збирання та аналізування текстової інформації, розроблення структури системи та її підсистем збирання та аналізування даних, налаштування процесу розробки, імплементування системи та її підсистем, тестування та верифікація роботи розробленої системи, розроблення стартап-проекту.
6. Перелік графічного (ілюстративного) матеріалу: діаграма послідовності створення запиту для збору та аналізу даних, діаграма класів доменної моделі, структурна схема системи, структурна схема підсистеми аналізу даних, структурна схема підсистеми

збору даних, ER діаграма доменної моделі, діаграма прецидентів, схема безперервної інтеграції та розгортання системи.

7. Орієнтовний перелік публікацій: _____

8. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання 07 вересня 2019

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Аналізування предметної області, аналізування існуючих рішень	16.09.2019	
2	Дослідження та аналізування методів збирання та аналізування текстової інформації	19.09.2019	
3	Розроблення структури системи та її підсистем збирання та аналізування даних	27.09.2019	
4	Налаштування процесу розробки, імплементування системи та її підсистем	28.10.2019	
5	Тестування та верифікація роботи розробленої системи	07.11.2019	
6	Розроблення стартап-проекту	19.11.2019	
7	Оформлювання дисертаційної роботи	05.12.2019	

Студент _____ Б. А. Погорілий
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ О. І. Ролік
(підпис) (ініціали, прізвище)

Пояснювальна записка

до магістерської дисертації

на тему: «Система збору та аналізу текстових даних з соціальних мереж»

Київ – 2019

РЕФЕРАТ

Магістерська дисертація: 102 с., 25 рис., 30 табл., 28 джерел.

Соціальні мережі та їх аналоги відіграють значну роль у сучасних умовах соціуму. Значно впливають на соціальні процеси інструменти, які об’єднують та реалізовують швидку та зручну взаємодію між суспільством. Різноманітні інциденту такі як спортивні змагання, фестивалі, президентські кампанії, вибори, мітинги чи революції не обходяться без обговорень та дискусій у соціальних мережах, мікро блогах та сервісах, де юзери можуть обмінюватись думками. Для прогнозування та ідентифікації подій, які очікуються або тих, які вже трапились, необхідно провести ретельний аналіз текстових даних, які постійно утворюються у мережі інтернет а саме, у соціальних мережах. Розв’язання цих задач вимагає розроблення методів та інструментів збирання та аналізування текстових даних отриманих з соціальних мереж та супутніх сервісів.

Метою дисертації є покращення швидкості, оперативності та мобільності систем збирання та аналізування текстових даних за рахунок розроблення та впровадження методів одержання текстових даних з соціальних мереж, а також їх попереднього оброблення, базового аналізування та агрегації.

Об’єктом дисертаційної роботи є процес оброблення та аналізування текстової інформації отриманої з соціальних мереж.

Предметом дисертації є методи моделювання, алгоритми потокової оброблення даних, методи попереднього оброблення текстової інформації, методи визначеннятафільтрування шумних текстових даних з використанням статистичних методів та інструментів оброблення натуральних мов, інструменти агрегації та збереження даних для їх використання аналітичними засобами.

У належності до поставленої мети, основні задачі дисертації полягають у розробці методів збирання та аналізування текстових даних, що включає у себе:

- налагоджування транспорту даних;
- попереднього опрацювання тексту;
- фільтрацію шумного контенту;
- знаходження сентиментів текстової інформації;

– агрегація отриманих результатів для подальшого використання аналітиками.

У магістерській дисертації виконано дослідження систем збирання та аналізування даних, проаналізовано існуючі рішення , з яких обрано оптимальні для використання у проектуванні та розробці. Проаналізовані, описані та реалізовані методи та алгоритми забезпечення швидкого та якісного збирання текстових даних.

В рамках дисертаційної роботи розробляється програмний комплекс, який складається з системи збирання та аналізу, а також графічного інтерфейсу юзера. Визначаються програмні компоненти, їх модель та архітектура, необхідні технічні інструменти для імплементації. Якість результатів аналізування покращується, при застосуванні методів фільтрації текстового шум.

Серверна частина є основним компонентом розроблюваної системи. Вона орієнтована на швидкий збір текстових даних з мікроблогового сервісу Twitter та їх якісний аналіз. Застосовуючи потокові методи та підходи інформація обробляється, як потік текстових даних для одержання можливості відображати результати аналізування у реальному часі.

Інтерактивний веб інтерфейс розробляється у якості графічного користувацького інтерфейсу для показу отриманих результатів аналізування представлених на масштабованих графіках з вказаною часовою лінією. Також юзерам доступна карта із зображенням географічних локацій активних юзерів.

В роботі представлена ER діаграма, діаграма прецедентів, діаграма послідовності, структурна схема системи та її підсистем, схема оброблення текстової інформації, алгоритм аналізування та схема безперервної інтеграції системи.

Теги: соціальні мережі, обробка натуральних мов, системи збирання даних, аналіз даних, текстовий шум

ABSTRACT

Master's thesis: 102 pages, 25 figures, 30 tables, 28 sources.

Social networks and their analogues play a significant role in today's society. The social processes are significantly influenced by the means that unite and realize the quick and convenient interaction between the society. Various events such as sports, festivals, presidential campaigns, elections, rallies or revolutions do not go unchallenged and discussed on social networks, micro-blogs and services where users can exchange views. In order to predict and identify what is expected or what is happening, it is necessary to carefully analyze the textual data that is constantly being generated on the Internet, namely, social networks. Addressing these challenges requires the development of methods and tools for collecting and analyzing textual data obtained from social networks and related services.

The aim of the dissertation is to improve the speed, efficiency and mobility of text data collection and analysis systems by developing and implementing methods for obtaining text data from social networks, as well as their pre-processing, basic analysis and aggregation.

The object of the dissertation is the process of processing and analyzing textual information collected from social networks.

The subject of the dissertation is modeling methods, algorithms of streaming data processing, methods of preliminary processing of textual information, methods of definition and filtering of noisy textual data using statistical methods and tools of natural language processing, means of aggregation and storage of data for their use by analytical means.

In accordance with the stated goal, the main tasks of the dissertation are to develop methods for collecting and analyzing textual data, which includes:

- adjustment of data transport;
- preliminary text processing;
- filtering of noisy content;
- finding sentiment of textual information;
- aggregation of results for further use by analysts.

In the master's thesis the research of the systems of data collection and analysis was performed, the existing solutions were analyzed, the optimal ones for design and

development were selected. Algorithms that provide text analysis are analyzed, described and implemented.

As part of the dissertation, a software complex is being developed, consisting of a system of collection and analysis, as well as a graphical user interface. The software components, their model and architecture, the necessary technical tools for implementation are determined. The quality of the analysis results is improved when text noise filtering techniques are used.

The server part is the main component of the developed system. It is focused on the rapid collection of text data from the microblogging service Twitter and quality analysis. Using streaming methods and approaches, information is processed as a flow of text data to enable real-time analysis of the results.

An interactive web interface is developed as a graphical user interface to display the results of analysis presented on scaled graphs with the specified timeline. A map depicting the active users' geographical locations is also available to users.

The paper presents an ER diagram, a precedent diagram, a sequence diagram, a structural diagram of a system and its subsystems, a scheme of textual information processing, an algorithm of analysis and a scheme of continuous integration of the system.

Tags: Social Media, Natural Language Processing, POS tagger, Data analysis system, Chi-Squared test, Textual Noise

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API (Application Programming Interface) – інтерфейс програмування застосунків

CD (Continious Delivery) – безперервна доставка

CI (Continious Integration) – безперервна інтеграція

DB (Database) – база даних

FTC (Federal Trade Commission) – Федеральна торгова комісія

HTTP (Hyper Text Transfer Protocol) – протокол передавання гіпер-текстових даних

NLP (Natural Language Processing) – оброблення натуральних мов

OOV (Out of Vocabulary) – позамовне слово

POS tagging (Part of Speech Tagging) – розмічування частин мови

SQL (Structured Query Language) – мова структурованих запитів

VCS (Version Control System) – система контролю версіями

VDS (Virtual Dedicated Server) – віртуальний призначений сервер

VPS (Virtual Private Server) – віртуальний приватний сервер

БД – база даних

ПЗ – програмне забезпечення

ВСТУП

Соціальні мережі та їх аналоги відіграють значну роль у сучасних умовах соціуму. Значно впливають на соціальні процеси інструменти, які об’єднують та реалізовують швидку та зручну взаємодію між суспільством. Різноманітні інциденту такі як спортивні змагання, фестивалі, президентські кампанії, вибори, мітинги чи революції не обходяться без обговорень та дискусій у соціальних мережах, мікро блогах та сервісах, де юзери можуть обмінюватись думками. Для прогнозування та ідентифікації подій, які очікуються або тих, які вже трапились, необхідно провести ретельний аналіз текстових даних, які постійно утворюються у мережі інтернет а саме, у соціальних мережах. Розв’язання цих задач вимагає розроблення методів та інструментів збору та аналізування текстових даних отриманих з соціальних мереж та супутніх сервісів.

Метою дисертації є покращення швидкості, оперативності та мобільності систем збору та аналізування текстових даних за рахунок розроблення та впровадження методів одержання текстових даних з соціальних мереж, а також їх попередньої обробки, базового аналізування та агрегації.

Об’єктом дисертаційної роботи є процес оброблення та аналізування текстової інформації отриманої з соціальних мереж.

Предметом дисертації є методи моделювання, алгоритми потокової обробки текстових даних, методи попереднього оброблення текстової інформації, методи визначення та фільтрування шумних текстових даних з використанням статистичних методів та інструментів обробки натуральних мов, інструменти агрегації та збереження текстових даних для їх використання аналітичними засобами.

У належності до поставленої мети, основні задачі дисертації полягають у розробці методів збору та аналізування текстових даних, що включає у себе:

- налагоджування транспорту текстових даних;
- попереднього опрацювання тексту;
- фільтрацію шумного контенту;
- знаходження сентиментів текстової інформації;
- агрегація отриманих результатів для подальшого використання аналітиками.

У магістерській дисертації виконано дослідження систем збору та аналізування текстових даних, проаналізовано існуючі рішення , з яких обрано оптимальні для використання у проектуванні та розробці. Проаналізовані, реалізовані та дослідженні методи та алгоритми забезпечення швидкого та якісного збору текстових даних.

В рамках дисертаційної роботи розробляється програмний комплекс, який складається з системи збору та аналізу, а також графічного інтерфейсу юзера. Визначаються програмні компоненти, їх модель та архітектура, необхідні технічні інструменти для імплементації. Якість результатів аналізування покращується, при застосуванні методів фільтрації текстового шум.

Серверна частина є основним компонентом розроблюваної системи. Вона орієнтована на швидкий збір текстових даних з мікроблогового сервісу Twitter та їх якісний аналіз. Застосовуючи потокові методи та підходи інформація обробляється, як потік текстових даних для одержання можливості відображати результати аналізування у реальному часі.

Інтерактивний веб інтерфейс розробляється у якості графічного користувацького інтерфейсу для показу отриманих результатів аналізування представлених на масштабованих графіках з вказаною часовою лінією. Також юзерам доступна карта із зображенням географічних локацій активних людей.

В роботі представлена ER діаграма, діаграма прецедентів, діаграма послідовності, структурна схема системи та її підсистем, схема обробки текстової інформації, алгоритм аналізування та схема безперервної інтеграції системи.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ОБРОБЛЕННЯ ТЕКСТОВИХ ДАНИХ

1.1 Існуючі рішення аналізування натуральних мов та їх недоліки

У магістерській дисертації досліджено аналіз текстових даних отриманих з соціальних мереж та додаткові аспекти їх оброблення. Значною мірою впливають і отримані результати аналізування вхідних текстових даних на якість отриманих вихідного результату. Емпіричні методи або статистично-навчальні машинні алгоритми застосовуються для обробки натуральних мов. Необхідно отримувати чи генерувати дані для розроблення або навчання чи тестування. Такі отримання текстових даних доцільно проанотувати.

Тестові дані важливо проанотувати, для оцінки алгоритмів аналізування. Анотаціювання тренувальних вхідних текстових даних важлива в зв’язку з тим, щоб відбувалося контролювання алгоритмів навчання, інакше навчальні алгоритми ймовірно працюватимуть мало ефективною тому, що мають змогу використовувати текстові дані, без допоміжних поміток, що явно впливає на якість аналізування. Проте випадковість одержання користі від неповноцінного анотованого набору текстових даних залишається [1].

Уникання спамового тексту у блогах та соціальних мережах під час збирання бажаних текстових даних – є ще одиним викликом, який доцільно розглядати та проаналізувати.

Присутня у соціальних мережах інформація, являється легкодоступною для кожного юзера, проте фігурує і приватна. Досліджено конфіденціальність користувацької інформації та того, як обсяг інформації доступної кожному може використовуватись.

Проблеми шумності текстових даних також присутні. Більша частина текстового контенту утворюється у важкому для аналізування форматі тому, що юзери здатні генерувати помилки, коли пишуть, спеціально писати незрозумілі текстові повідомлення. Тому важливо зосередити увагу на обробці чи фільтруванні такого потоку текстових даних за для одержання кращих результатів аналізування та обробки інформації.

Досліджено подібні проекти для аналізування текстових даних з мікроблогового сервісу Twitter . Підвищення популяризації мікро блогів призвело до підвищення масштабів досліджень. Досліджено географічний та топологічний розподіл мікроблогової мережі Twitter. Встановлено, що твітер повідомлення демонструють властивості соціальної мережі більше, ніж інформаційні та фактологічні матеріали, наприклад, соціальні мережі. Детально розмежовано інформаторів, які поширюють інформацію, та само інформаторів, які обговорюють свої власні справи. Описано аналіз використання мікро блогів та інформаційних життєвих циклів під час кризових ситуацій [2].

Актуальна галузь досліджень мікро блогів аналізовує реакції на новинні інциденту у мікроблоговому сервісі Twitter. Велика кількість статей даного розділу розглядає утворювання рукописних візуалізацій та графіків, досить схожих до тих, які утворені системою інтерактивно та автоматично. Зроблена дослідницька робота у даному сегменті, яка показовує використання сентиментальних метрик шкали часу для вивчення дебатів президентів у 2008 році через мікроблоговий сервіс Twitter [2]. Також продемонстровано подібне детальне аналізування під час інавгурації президента Обами, з'ясовуючи, що обсяги твітер повідомлень підвищуються і результати спадають у час потрібних моментів. Спостерігання на часовій шкалі за твітер повідомленнями з різноманітних географічних районів, коли відбувається повені на річці Червона. Виконано деталізоване аналізування недавніх виборів у країні Іран, у якому мережа Twitter відіграла основну роль у новини звітах.

Виявлено, що аналіз емоцій може дати корисну інформацію про продукт [2]. Метою магістерської дисертації є розроблення та утворення таких аналізувань швидшими та легшими, де знання з програмування не вимагаються.

Система базується на нещодавному дослідженні присвяченому тимчасовому аналізуванні з використовуванням інтерфейсів пошуку, які доволі близько пов'язані з роботами систем аналізування, які разом забезпечують графічні візуалізації текстових даних мікроблогового сервісу Twitter [3]. Дані системи фокусуються на подіях медіа, а системи одиничного збирання текстових даних – тільки на одному каналі.

Також досліджено інакшу роботу, що розглядає часове візуалізування. Системи аналізування часових подій відслідковують відомі фрази через новинні ЗМІ та блоги, а не систему Twitter. Система аналізування текстових даних «зоотроп» відображає юзереві графіки тенденцій веб-контенту, як переживання та емоції людей [4].

Необхідно розглянути деякі пов'язані не академічні проекти. Наприклад, Sentiment Twitter– це веб сайт, який показовує періодичність повідомлень на шкалі часу, коди та кругову діаграму емоцій, що налаштовуються на основі свідомості. З сайту запозичено багато візуальними елементів, а також використано їхній підхід до виявлювання сентиментів твітер повідомлень [5].

Google Timeline Новини покращують виучування носіїв новин у вказаних часових проміжках, але безпосередньо не відображають юзереві, які елементи використані при візуалізації даних. Системи подібні до Flipboard, перетворюють новини у соціально важливі, постачаючи подібний до журналу графічний інтерфейс для визначення новинних джерел, отриманих із користувацької соціальної мережі, наприклад, Facebook чи Twitter. Система Twitter збирається розбудувати систему аналітики у реальному часі, але такий інструментарій, схоже, не є спрямованим на інформування кінцевих юзерів системи або передавання новин [6].

1.2 Постановка функціональних вимог системи

Система збирання та аналізування текстових даних з соціальних мереж необхідна виконувати завдання спілкування з юзерами для забезпечування доцільного рівня спрощеної взаємодії. при реєстрації у платформі Messenger Facebook вказується персональна інформація юзера і відкрита для зчитування тільки системі аналізування та юзеру під час взаємодії, шляхом інтегрування з програмним прикладним інтерфейсом мережі Facebook.

Розроблювана система націлена на користувацьку аудиторію, яка хоче отримувати дані та актуальну інформацію про інциденту та новини, які є активно обговорюваними у соціальних мережах. До таких юзерів можна віднести пересічних

громадян, а також персон, які потребують у ході своєї професійної діяльності оперативну інформацію про новини та інциденту, а саме, блогери чи журналісти.

Проаналізувавши розроблені системи, виявлено та доведено їх основні функціональні недоліки та переваги, що дозволяє описати та виділити вимоги до системи збирання та аналізування даних. Також встановлено функціональні базові вимоги до системи:

- утворювання запиту;
- аналізування запиту, методами оброблення текстових даних отриманих з Twitter;
- перегляд аналізування з використанням графічного інтерфейсу;
- відображування часового графіку з піковими точками;
- масштабування активності юзерів на часовому графіку;
- вибір розмірності вікна часу;
- відображування активних юзерів як точок локацій на карті;
- одержання стану запущену систему.

1.3 Збережування текстових даних та їх анотація

Величезна кількість користувальницького веб-вмісту та зростаюча популярність соціальних мереж утворюють умови для отримування доступу до текстових даних загально доступного характеру. Хоча анотація та збір таких текстових даних з мікро блогів чи сервісів онлайнових та інших соціальних мереж становлять проблему для програм оброблення натуральної мови.

1.3.1 Збір текстових даних із соціальних мереж

Збір соціальних текстових даних у основному залежний від визначеного алгоритму та завдання. З соціальних мереж текстові дані зазвичай збираються у різноманітних формах, таких як мікро блогові повідомлення, описи зображень, коментування повідомлень, метадані та відеоролики. Також є цікавим взаємо

з’єднання текстових даних, наприклад, зв'язок між платформами соціальних мереж, наприклад, мережа Instagram та мікро блог Twitter, або зв'язування твітер повідомлень з новинами.

API, а тобто програмний прикладний інтерфейс соціальних мереж надає можливість інтегрування інших сервісів з їх платформами, в зв’язку з тим для одержання доступу до всієї кількості заздалегіть отриманих та агрегованих текстових даних та метаданих, таких як інформація про людей, їх адреса, стать, дата народження, контактитаінше. Такими інтерфейсами можна назвати Facebook API чи Twitter Stream API.

Проте збір текстових даних із соціальних мереж має певні обмежування. Наприклад, служби мікроблогування, такі як Twitter, мають обмежування швидкості API для кожної програми або для кожного юзера. Це задовольняє обмеження швидкості та кількості запитів. Утворено оплачуваний доступ для підтримання зчитування збільшеного об’єму повідомлень за одиницю часу, для більшої кількості користувацьких текстових даних мережі Twitter [7].

1.3.2 Анотація тексту з соціальних мереж

Анотування текстових даних отриманих з соціальних мереж є складним завданням. Анотаційне завдання можна виконати напівавтоматично, застосовуючи попередньо запрограмовані інтерфейси.

Наприклад, зручним інструментом для анотації є TwitIE – компонент соціальної мережі та GATE – загальна архітектура для текстової інженерії. автоматично генерувати теги анотацій намагалися деякі дослідники для позначення інтересів юзерів мігроблогового сервісу Twitter, застосовуючи такий рейтинг тексту, як TextRank та FTF-ID, моделі яких засновані на графах класифікації тексту для видалення з твітер повідомлень цільових слів, щоб ідентифікувати активного юзера. Для анотації виборки текстових даних Twitter з аргументаційними класами, дослідники застосували машинне контрольоване навчання. Існують випадки, коли анотування своїх повідомлень займаються безпосередньо юзері. Під час машинного

навчання часто використовують такі мітки. Як приклад – це стікери емоцій на платформі LiveJournal [8].

1.3 Ідентифікація шуму та спаму

Коли мова йде про вибір ресурсів для збирання та аналізування текстових даних у соціальних мережах, сканування мільйонів щоденних соціальних бесід або простий акт прослуховування не є достатнім. У соціальних мережах світу з мільйонами активних юзерів, об’єм користувальницького контенту зріс значно. Обсяги твітер повідомлень виросли з 15 тисяч щоденних у 2012 році до 500 мільйоні твітер повідомлень у день у 2018 році, і він зберужує свою популярність і до цього часу. Найважливішим є вибір стратегії, яка найліпше підтримує показники та цілі системи, при застосуванні необхідних аналітичних методів та підходів на основі обробки натуральних мов NLP. Значна кількість шумів та спаму у соціальних мережах викликала обговорювання цінності та обґрунтованості текстових даних отриманих з соціальних мереж, якщо такі дані дуже залежні від часу та місця [9].

Новітні інструменти спілкування, починаючи від соціальних мереж та закінчуючи текстовими повідомленнями, кардинально змінюють методи та інструменти мовного використання.

Нові лінгвістичні формування доцільно використовувати у мережі Інтернеті, наприклад, багато перських юзерів або юзерів Північної Африки та Близького Сходу, застосовуючи латинський символ зі схожою вимовою словосполучення в оригіналі – яка є однією з транслітераційних форм, висловлюють свої погляди та думки у соціальних мережах та мікроблогах.

В англійській мові надто популярними стали такі словосполучення та скорочення, як TTYL – поговорю з тобою пізніше, OMG – о, мій Бог та LOL – сміятися голосно. Деякі новоутворені словосполучення навіть вдалось ввести у словники, наприклад, селфі – іменник, у 2013 році та як ретвіт – дієслово, у 2011 році які успішно додані до словника Кембріджу.

Існують різноманітні причини шуму текстових даних з соціальних мереж, які можуть викликати перешкоди для оброблення натуральних мов інструментами, такими як машинний переклад, виявлювання думок та пошук інформації. Наприклад, неправильно написані словосполучення завжди присутні у соціальних мережах. Завдання та призначення нормалізації може частково визначати помилки мови належно до наведених фактів.

Застосовуючи статистичне та лінгвістичне аналізування до корпусів, отриманих у соціальних мережах, T. Catarci проаналізував шум текстових даних отриманих з соціальних мереж, і порівняв їх з контрольними тестовими даними. Проведено аналізування пропозицій, відносне відчуття слів поза словником та лексичну композицію, щоб побачити, як вивчаються неграматичні регіональні лінгвістичні варіації шляхом моделювання зв'язку між географічними регіонами та словами. Доведено, що для вибраних тематик, таких як спорт чи погода жаргони та фразеологізми подаються різним чином залежно від регіону. Як приклад, тему, пов'язану зі спортом, доцільно проаналізувати по різному у Каліфорнії та Нью-Йорку [9].

Відомий факт, що текст у соціальних мережах є «шумними». Шумний текст – це текстові дані, які не можна правильно обробити з використанням програмного забезпечення для оброблення тексту. Докладено недостатньо зусиль для кількісної оцінки того, наскільки текст у соціальних медіа є шумніший, за звичайний відредагований текст. Також, текстові дані у соціальних мережах мають різноманітні представлення, наприклад, мікро блоги, блоги чи коментарі, утворювані юзерами.

Досліджування, як правило, зосереджуються на конкретному джерелі текстових даних, наприклад, твітер повідомлення або коментарі з вибраного ресурсу. Закономірне запитання – як відрізняються текстові дані отримані з різноманітних джерел. Визначення шумності тексту є важливим першим кроком до побудови набору інструментів загального призначення для оброблення текстових даних отриманих з соціальних мереж. Проведено аналізування, яке кількісно визначає, наскільки важко застосовувати NLP до тексту соціальних медіа.

1.3.1 Визначення основної мови тексту

В зв’язку з тим, щоб оцінити характеристики тексту з різноманітних джерел, отримано набори текстових даних із усього спектру популярних сайтів соціальних медіа, що залежать від довжини документа, рівня редагування тексту та кількості авторів на документ. Отримано твітер повідомлення з мережі Twitter за два періоди, коментарі з Youtube, статті з Wikipedia, блог пости з Spinn3r, а також документи з the British National Corpus, далі – BNC. Набір текстових даних описано у таблиці 1.1.

Таблиця 1.1 – Кількість документів та середній розмір документа для кожного набору текстових даних

Джерело	Кількість документів	Середня кількість слів у документі
Twitter-1	1000000	11,8
Twitter-2	1000000	11,6
Youtube	874772	15,8
Blogs	1000000	147,7
Wikipedia	200000	281,2
BNC	3141	31609,0

Проаналізовано розбиття мов, знайдених у кожному джерелі текстових даних, виходячи з результатів langid.py. Отримані результати базуються на повних наборах даних без мовної фільтрації. У всіх джерелах даних переважає англійська мова.

1.3.2 Попереднє оброблення та лексичне аналізування тексту

Спочатку попередньо обробляється кожен набір даних, застосовуючи стандартизовану методологію Natural Language Processing, далі – NLP [10]. У випадку, якщо текстові дані постачаються з токенізацією та інформацією про Part of Speech теги, далі – POS теги, видаляються та виконується автоматична попереднє оброблення, щоб забезпечити узгодженість якості та складу тегів.

У таблиці 1.2 представлено результати аналізування лексичного складу англійських документів, а саме статистику щодо середньої довжини слова, які виміряно у символах, та середньої довжини речень, які виміряно у словах, для кожного набору даних. Також проаналізовано відносне виникнення позамовних слів Out of Vocabulary, далі – OOV, на основі словника GNU Aspell v0.60.6.1 з ігноруванням регістру символів [11].

Таблиця 1.2 – Середня довжина словосполучення та речення та частка слів OOV у кожному наборі текстових даних

Джерело	Довжина слова	Довжина речення	OOV %	
			-norm	+norm
Twitter-1	3,8	9,2	0,246	0,225
Twitter-2	3,8	9,0	0,240	0,222
Youtube	3,9	10,5	0,198	0,184
Blogs	4,1	18,5	0,206	0,203
Wikipedia	4,5	21,9	0,190	0,188
BNC	4,3	19,8	0,196	0,168

Для зменшування шумності тексту знайдено та видалено всю «онлайн» розмітку, таку як хештеги, посилання на юзерів та URL-адреси, з використанням POS-тегера. Щоб відфільтрувати поширені сленгові вирази, такі як «ur», використано попередній крок «лексичної нормалізації» на основі словника Хана, який дає стандартну форму для даного OOV, засновану на комбінованій інформації зі сленгових словників та автоматично отриманих належностей «-norm», «+norm».

1.3.3 Граматичне аналізування

Наступним логічним питанням є те, наскільки граматично правильним є текст у кожному з наборів даних. Вимірюється граматичність з використанням англійської ресурсної граматики – ERG [12]. У таблиці 1.3 наведено результати розбирання 4000

випадково вибраних англійських речень з кожного текстового корпусу з використанням ERG.

Таблиця 1.3 – Відсоток речень, які можна проаналізувати з використанням ERG, розбиті на кореневу умову синтаксичного розбирання

Джерело	Піддаються граматичному аналізуванню				Не піддаються граматичному аналізуванню
	формальний		неформальний		
	повн.	частк.	повн.	частк.	
Twitter-1	13,8	23,9	22,2	2,5	37,4
Twitter-2	13,9	23,8	22,8	1,7	37,6
Youtube	18,0	22,2	26,4	1,4	31,9
Blogs	25,6	14,1	24,7	2,7	35,6
Wikipedia	48,7	4,1	18,8	1,5	26,2
BNC	38,4	12,2	24,0	2,2	23,2

Одним з аспектів ERG, який робить його дуже придатним для перевірки граматичності, є те, що, на відміну від більшості парсерів NLP, він є «генеративним», тобто чітко моделює граматичність, і розроблений відносно як позитивних, так і негативних тестових елементів, щоб гарантувати якісне аналізування. В зв’язку з тим, щоб максимально охопити лексичне покриття ERG, застосовуються POS-обумовлені загальні лексичні типи.

1.3.4 Подібність текстових даних

Досліджування внутрішньої подібності за стилем та змістом різноманітні наборів даних потребує використання статистичного аналізування (таблиця 1.4).

Таблиця 1.4 – Парна подібність текстів з різноманітних ресурсів за результатами тесту хі-квадрат

	Twitter-2	Youtube	Blogs	Wikipedia	BNC

Twitter-1	4,0	63,7	91,8	347,8	251,8
Twitter-2	-	62,4	90,6	360,0	258,8
Youtube	-	-	128,4	351,4	245,2
Blogs	-	-	-	157,7	78,7
Wikipedia	-	-	-	-	92,5
BNC	-	-	-	-	-

Один з можливих методі полягає у обчисленні «схожості тексту» між наборами даних та однорідності у межах даного набору даних. у дослідженні [12] вимірювання подібності та однорідності тексту запроваджено метод, заснований на тесті хі-квадрата, з використанням якого вимірюється подібність двох текстів, як статистика тесту хі-квадрат на 500 найчастіших слів у об’єднанні текстів, який використано у дослідженні.

1.4 Ідентифікація дійсної інформації

В ході інформаційної перевірки даних з соціальних мереж необхідно довіряти та знати джерело. Головним завданням являється знаходження оригінального та правдивого джерела, яке можна перевірити з використанням аналізування соціальної мережі та методів обробки натуральних мов NLP, які вивчали дані про походження та легко ідентифікувати, пов'язані з текстовими повідомленнями отриманими з соціальної мережі, що може допомогти спростувати плітки, підтверджувати факти та з'ясувати думки. У магістерській дисертації описано три етапне аналізування атрибутики походження текстової інформації, а саме генерація через мережеву інформацію та знаходження текстової інформації про походження даних, з використанням мережевих даних. Окремим завданням є збір текстової інформації, яка містить достатньо даних, а не збір текстової інформації без корисного навантажування. Проведено намагання визначити інформативність повідомлень у соціальних мережах та класифікувати їх, виходячи історії юзера або з тексту, особливо у текстових даних отриманих з мережі Twitter [12].

Онлайн-блоги та соціальні медіа утворюють умови коментування та задавання питанньдля читачів.

Надто важливо, щоб користувальники реагували на відгуки та думки у мережі Інтернеті для різноманітних цілей, таких як покупка послуг або нових продуктів, пошук готелю чи ресторану або консультування з психологом чи терапевтом. Тому важливим маркетинговим інструментом стали огляди коментарів. Позитивні відгуки призводять до неймовірного успіху цільових осіб чи окремих організацій та підприємств. На превеликий жаль, така ситуація також привертає думки про спам та підроблені повідомлення соціальної мережі, які вводять у оману автоматизовані системи семантичного аналізування або юзери мережі.

Фальшиві «негативні» відгуки та підроблені «позитивні» огляди мають значне впливання. Виявлювання достовірність думок або спаму у мережі Інтернет можуть допомогти знаходження фальшиві негативні чи позитивні відгуки, за для уникнення репутаційного пошкодження, а також уникнення одержання спаму та шумного контенту з соціальної мережі у ході навчання та збирання текстових вхідних даних. Досліджування показали, що від 15 до 45 відсотків коментарів Yelp являються фальшивими. Маніпулювання фальшивими думками та онлайн-відгуками, більш за все, Федеральна торгова комісія – FTC признає нелегальним у рамках закону про захист прав споживачів.

Вивчення проблеми обману, було досягнуто шляхом навчання моделей, з використанням контрольованого навчання, з використанням посібника навчання з ручним анотуванням даних, на основі продукту, тексту огляду та оглядача для виявлювання фальшивих думок та коментарів. Вивчено проблему ідентифікації дистриб'юторів компанії у Twitter, утворюючи рекламу для певної цільової продукції, ідеї, повідомлення чи послуги. Також проаналізовано різницю між правдивими відгуками на основі набору текстових даних про золотий стандарт, який складається з текстових даних з трьох різноманітних областей – готелі, ресторани та лікарні та використанням мовлення між хибними повідомленнями у Інтернеті [13].

Інше джерело спаму утворюється спам ботами. Спам бот являється комп'ютерною автоматизованою програмою, яка розроблена для відправлення спаму

та фальшивої інформації. Такі системи зазвичай працюють від імені фейкових облікових записів у певній соціальній мережі та розсилають спамову інформацію з використанням них.

1.5 Висновки до розділу 1

У розділі отримано та опрацьовано текст з різноманітних джерел, а саме мікро блогів, утворених юзерами коментарів, блогів, наукових ресурсів. Застосовано лінгвістичне та статистичне аналізування, а саме ідентифікування мови, лексичне аналізування, граматичний та методи визначення подібності тексту.

Проаналізовано «шумність» текстів соціальних медіа і показано, що методи NLP, включаючи ідентифікування мови, лексичне нормалізування та POS тегування можна застосувати, щоб зменшити шумність текстових даних. Це доводить, що згенерований юзерами контент насправді шумний, але з використанням інструментів NLP він очищається і застосовується для подальшого якісного аналізування.

З точки зору граматичності, отримані результати підтвердили, що текст соціальних медіа менш граматичний ніж відредагований текст, але розбіжність порівняно невелика.

Аналізування подібності тексту показав, що проаналізовані типи текстів соціальних мережах, лежать у континуумі подібності. Цей висновок має потенційні наслідки для вибору навчальних текстових даних для статистичних систем NLP.

2 ВИБІР МЕТОДІВ ОБРОБЛЕННЯ НАТУРАЛЬНИХ МОВ

2.1 Аналізування технічних засобів оброблення натуральних мов

Оброблення натуральних мов включено до багатьох фреймворків текстового оброблення. Деякі з них обмежені у функціональності, але доступні у Java SE SDK. Доступна тільки функціональність розв’язання найпростіших типів проблем. Проте, інші бібліотеки надають підтримку різноманітних інструментів, такі як LingPipe OpenNLP та Apache.

Пакет Java включає бібліотеку, яка містить StringBuffer, StringBuilder і String. У названих класах доступні методи, які забезпечують знаходження заміну і узгодження тексту. Для співставлення використовують регулярні вирази зі спеціальним кодуванням. Для використання регулярних виразів Java інструментарій надає велику кількість методів.

Для розбиття тексту на окремі елементи застосовуються токенізатори. У Java підтримка токенізаторів реалізована у наступних інструментах:

– клас String та його методи розбиття;

– StreamTokenizer клас;

– BasicStringTokenizer клас.

Наприклад, зручно розбивати заданий текст на токени з використанням StreamTokenizer (рисунок 2.1).

```
FileReader fileReader = new FileReader("../test.txt");
StreamTokenizer fileTokenizer = new StreamTokenizer(fileReader);
while (fileTokenizer.nextToken() != StreamTokenizer.TT_EOF) {
    if (fileTokenizer.ttype == StreamTokenizer.TT_NUMBER) {
        System.out.println(fileTokenizer.nval);
    } else if (fileTokenizer.ttype == StreamTokenizer.TT_WORD) {
        System.out.println(fileTokenizer.sval);
    }
}
```

Рисунок 2.1 – Приклад розбивання вхідного тексту на токени

З прикладу видно що файл зчитується з використанням інтерфейсу FileReader, який застосовується класом InputStreamReader, для прочитання текстового файлу з пам’яті середовища розгортання, для одержання токенів з вхідного тексту.

В відкритому доступі наявна велика кількість Java інтерфейсів та бібліотек обробки натуральних мов NLP. Неповний список найпопулярніших NLP інтерфейсів та API розроблених на основі мови Java зображено у таблиці 2.1. Такі бібліотеки є open source компонентами. Окремо від них доступнітакомерційні інтерфейси. Далі доцільно зосередити увагу на API з відкритим кодом.

Таблиця 2.1 – Перелік існуючих засобів оброблення натуральних мов

API	**URL**
Apertium	http://www.apertium.org/
General Architecture for Text Engineering	http://gate.ac.uk/
Learning Based Java	http://cogcomp.cs.illinois.edu/page/software_view/LBJ
LinguaStream	http://www.linguastream.org/
LingPipe	http://alias-i.com/lingpipe/
Mallet	http://mallet.cs.umass.edu/
MontyLingua	http://web.media.mit.edu/~hugo/montylingua/
Apache OpenNLP	http://opennlp.apache.org/
UIMA	http://uima.apache.org/
Stanford Parser	http://nlp.stanford.edu/software

Щоб утворити комплексну та складну послідовну обробку текстових даних, більшість з наведених інструментів NLP об'єднані. Для цього утворюються послідовності так звані пайплайни різноманітних завдань з використанням обробки натуральних мов NLP для досягнення ефективного оброблення, які інтегруються у низку кроків. Прикладами фреймворків, які підтримують пайплайни, є Apache UIMA і GATE.

2.2 Фреймворк Open NLP

Фреймворк Open NLP Apache – це комплект інструментів для оброблення тексту натуральної мови основаного на машинному навчанні. Даний фреймворк розв’язує найпопулярніші задачі обробки натуральних мов NLP, такі як сегментація пропозицій, маркування тегами частин мови, видалення названого об'єкта, токенізація, визначення корелювальної якості, синтаксичне аналізування та фрагментування. Такі задачі, як правило, застосовуються для утворення більш складних інструментів оброблення текстових даних. Open NLP підтримує машинне навчання на основі персептрону та максимальну ентропію.

Метою проекту Open NLP Apache заключається у утворенні комплексних інструментів для розв’язання задач обробки натуральних мов згаданих вище. Додатковою ціллю є постачання великого обсягу завчасно утворених моделей різноманітних натуральних мов, а також проанотованих та підготованих текстових семплів, з використанням яких утворені моделі.

Фреймфорк Open NLP Apache складається з кількох компонентів, які забезпечують утворювання повноцінного пайплайну для оброблення натуральної мови. До таких компонентів можна віднести: аналізатор, тегер, токенайзер, детектор виправлень, категоризатор документів.

Модулі фреймворку дозволяють вирішувати належні поставлені завдання оброблення натуральної мови, оцінювати модель та тренувати модель.

Кожен з таких модулів доступний через прикладний програмний інтерфейс фреймворку. Окрім цього, для зручності надається інтерфейс командного рядка CLI щоб проводити тренування та експерименти.

Модулі фреймворку Open NLP Apache розроблені таким чином, для виконання ряду задач необхідно задати вхідні дані і модель.

Щоб завантажити модель, необхідно використати FileInputStream клас. Наприклад, утворену модель необхідно задавати як параметр у методи інструментів бібліотеки, як це виконано з TokenizeME класом (рисунок 2.2).

```
InputStream is = new FileInputStream("en-token.bin");
TokenizerModel model = new TokenizerModel(is);
Tokenizer tokenizer = new TokenizerME(model);
String tokens[] = tokenizer.tokenize("../text.txt");
is.close();
Arrays.stream(tokens)
        .forEach(System.out::println);
```

Рисунок 2.2 – Приклад одержання токенів з тексту

2.3 Фреймворк Core NLP Stanford

Core NLP Stanford надає набір засобів аналізування натуральних мов, утворених на мові Java. Фреймворк може оброблювати інформацію представлені у текстовому форматі та як результат видавати частини мови форми, їх базові слів, будь то назви людей чи компаній. Також фреймворк нормалізовує та інтерпретує цифрові величини, час та дати. Вираховує реченнєву структуру з точки зору словосполучень або фраз, які фрази чи словосполучення стосуються тих чи інших форм мови.

Фреймфорк Core NLP Stanford утворено для обробки англійської мови, проте зараз він постачає різноманітні модулі підтримки мов таких, як материковова китайська, сучасний стандарт арабської мови, німецької, іспанської та французької.

Фреймфорк Core NLP Stanford – це інтегрована структура, яка забезпечує дуже легке використання різноманітних інструментів аналізування натуральних мов. Починаючи з простого тексту, можливий запуск всіх інструментів тільки з двома рядками коду. Даних фреймворк являється базовим інструментарієм для комплексних систем аналізування у яких необхідна реалізація розуміння текстових даних на вищому домені та рівні.

Фреймворк Core NLP Stanford – це агрегація добре протестованих та стабільних засобів оброблення натуральних мов, широко застосовуваними різними прошарками у промисловій, урядовій сферах та академічній. Інструменти широко використовують компоненти глибокого навчання, імовірнісний механізм навчання та правила обробки натуральних мов.

Код фреймоврку Core NLP Stanford ліцензований під загальною ліцензією GNU версії 5 чи старшої версії та розроблено на базі мови Java. Згадана ліцензія являється цілком повною версією GPL. Вона не дозволяє використовувати у комерційних проектах даний фреймворк, проте цілком легально застосовувати фреймворк у навчальних чи open source проектам.

Отже, на основі аналізування існуючих рішень оброблення натуральних мов вирішено, що доцільно застосувати фреймворк Open NLP Apache у розробці, так як умови використання Core NLP Stanford не передбачають використання фреймворку у комерційних проектах. Також прикладний програмний інтерфейс фреймворку Open NLP Apache набагато швидше інтегрується та є більш зрозумілішим для розробників.

2.4 Клієнт-серверна архітектура системи

Для імплементації системи доцільно застосувати саме клієнт-серверну архітектуру, яка являється широковживаним архітектурним рішенням (рисунок 4.1).

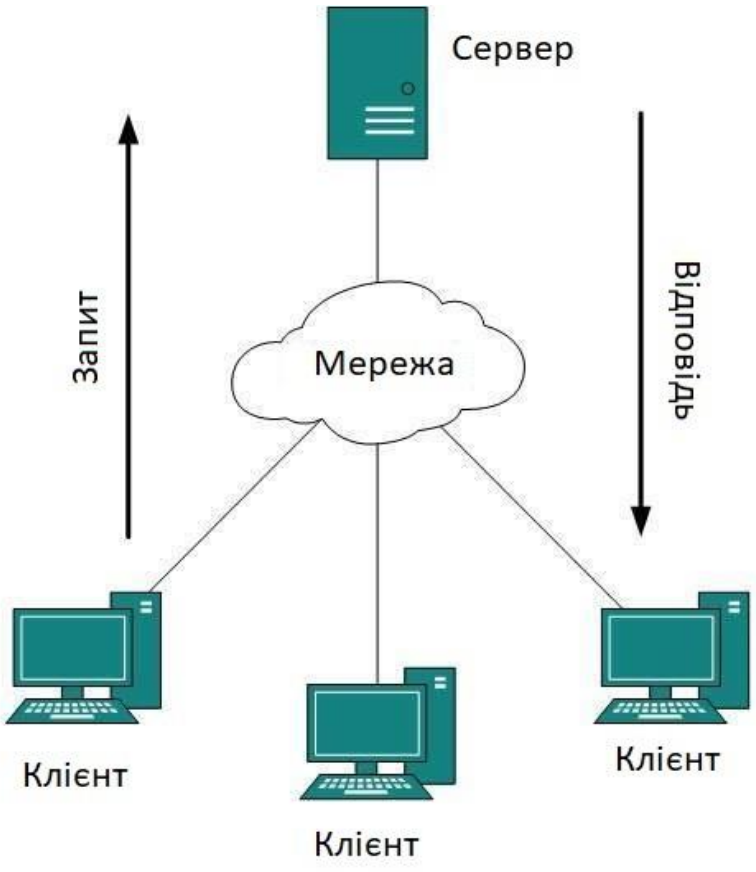

Рисунок 2.1 – Клієнт-серверна архітектура

Детальна архітектура системи представлена у додатку, де зображено основні компоненти та їх взаємодія.

Даний підхід доцільно вибрати тому, що виходячи з технічного завдання, важливо реалізувати інтерфейс для взаємодії з юзерами, який доступний кожному. Тому доцільно вибрати розробку веб інтерфейсу, який отримуватиме інформацію безпосередньо з централізованого сервера.

Належно до встановлених фактів клієнтська сторона реалізовуватиметься у вигляді веб-інтерфейсу та чат бота на платформі Messenger Facebook.

Для виконання обробки запитів юзери та клієнтів та відправки відповідностей на них, необхідно утворити централізовану систему спілкування з веб клієнтом. Як серверну складову даного підходу саме цю частина програмного комплексу доцільно використати.

Два базових модуля сервер та клієнт зб’єднані у єдину мережу та спілкуються з використанням протоколів обміну текстової інформації. Тому для цього доцільно вибрати протокол HTTP та мережу Інтернет.

Для імплементації серверної складової та клієнтської доцільно розглянути комплекс технічних засобів імплементації.

2.2 Трирівнева модель архітектури системи

Для імплементації серверної частини доцільно застосувати підхід три рівневого розбиття компонентів програми, таких як репозиторії, сервіси , та контроллери (рисунок 2.2).

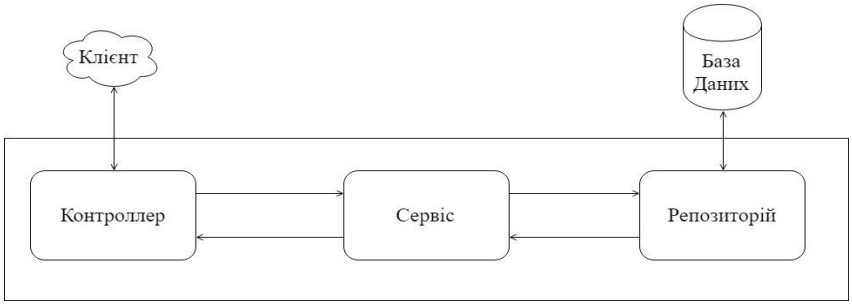

Рисунок 2.2 – Схема трирівневої моделі архітектури системи

За призначені функції відповідає кожен ступінь. Дані рівні спілкуються тільки через визначені спеціальні інтерфейси та є ізольованими одне від одного.

Як основний інструмент розроблення перш за все треба обрати фреймворк, що постачає легкий для впровадження та розуміння інструментарій, щоб реалізувати обраний архітектурний підхід.

За взаємодію та спілкування із зовнішніми системами відповідає ступінь контроллерів. На даному рівні описано програмний прикладний інтерфейс та імплементовано логіку системи безпеки. Такий ступінь відповідає за оброблення запитів сторонніх клієнтів та юзері, а також оперує рівнем сервісів для обробки отриманих даних.

Дані, які надходять з рівня контроллерів, ступінь сервісів аналізовує та оброблює. Саме на цьому рівні імплементовано всі логіки та алгоритми оброблення даних. Для читання та зберігання даних та інформації ступінь сервісів звертається до рівня репозиторіїв.

Десеріалізування та серіалізування текстових даних запитаних чи отриманих з сервісного рівня реалізовує ступінь репозиторіїв. На даному рівні проходить взаємодія з базою даних, яку було обрано не реляційною. Така база даних задовольнить поставлені вимоги до розроблюваної системи тому, що доменна модель системи є не комплексною і вимагає швидкого зчитування та запису даних до бази даних.

2.3 Потокове оброблення текстових даних

Для швидкого і безперервного оброблення текстових даних метод «належь запит» використовувати не можливо тому, що даний підхід є блокуючим для серверного модуля. Тому для досягнення бажаного результату доцільно обрати потокові технології транспорту та оброблення текстових даних.

Для задоволення бізнес вимог так звані пайплайни будуватимуться. Вони являтимуться послідовністю простих та комплексних оброблювалинків інформації через, які у протікають дані певній послідовності. Такий детермінованого оброблення

потік, є незалежним від будь-яких факторів зовнішніх чи внутрішніх. Доцільно зрозуміти, що таблиці і потоки – це, одне і те ж саме явище у даному контексті. Таблицю можна інтерпретувати як потік, а потік– як таблицю. Тому обраний метод допоможе безпробемно керувати даними при їх аналізуванні та обробці.

2.4 Структура проекту

Для імплементації проекту важливо вибрати найбільш прийнятний підхід компонування і проектування проекту, що посприяє його швидкому та якісному імплементуванні. Обрана структура повинна задовольняти зручні умови тестування, розгортання та масштабування у подальших етапах розробки.

Доцільно розглянути різноманітні підходи до проектування. Один з них це багатомодульний. Проект розподіляється на декілька модулів що, у реалізованому варіанті містить два – це і є суттю даного підходу. Модулі залежать від одного батьківського проектного модуля. Такі модулі іменують нащадками.

На рисунку 2.3 зображено модулі проекту, які виділені шрифтом жирним. у представленій імплементації батьківський модуль– це twitter-stream-analysis, який має у собі два вкладених модулі нащадки twitter та messenger.

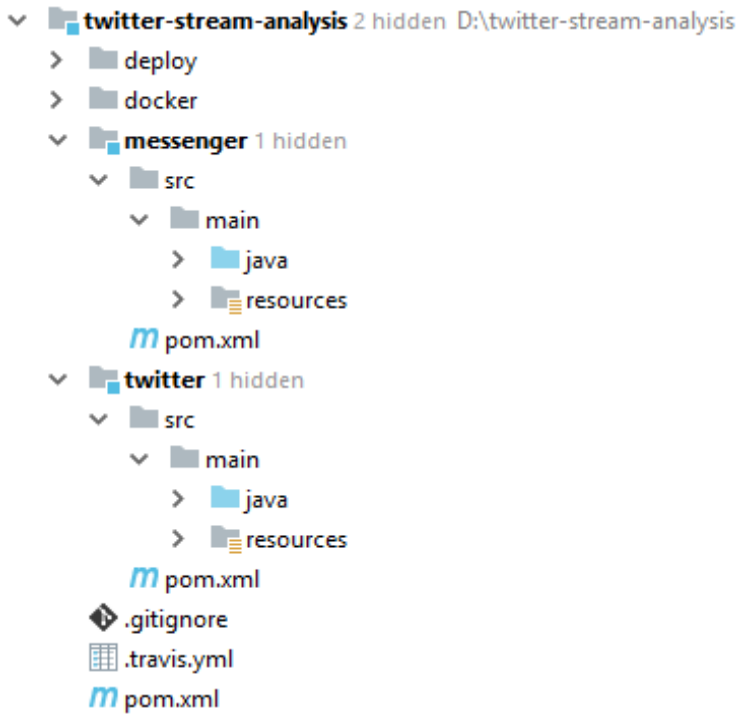

Рисунок 2.3 – Структура проекту

Належно модулі похідні включають власні конфігурації та налаштування, а батьківський утворює узагальнене системне конфігурування, а також описує збирання проекту логіку та репозиторії бібліотек.

Проект також включає директорії із файлами для пакування системи у контейнери – docker директорія, а також системного розгортання на хостингу хмарному – deploy директорія.

В корені проекту файли .travis.yml, .gitignore необхідні для розгортання зовнішніх сервісів та описані у розділі.

2.5 Висновки до розділу 2

Отже, у даному розділі сформовано та задану загальну структуру проекту та його компонентів. Обрано архітектуру, що дозволяє вибрати необхідні технології для імплементації поставленої задачі. Основуючись на результатах аналізування можна приступати до імплементації проекту та перш за все вибору технологічного стеку проекту.

Вибір технічних засобів та їх використання у імплементації, а також підходи, які використовувались для досягнення бажаного результату описано у наступному розділі.

3 ВИБІР ТЕХНІЧНИХ ЗАСОБІВ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Мова програмування Java

Java була утворена командою у Sun Microsystems, була випущена у 1995 році і згодом була придбана Oracle.

Основна мета творців Java - розробити мову, яка може працювати на побутовій техніці - тому дизайнери вже замислювались про світ, у якому на вашому холодильнику чи тостері з'явився код - те, що ми сьогодні називаємо Інтернетом речей. Ми тільки нещодавно почали генерувати пристрої, які користуються такою мовою, тож у середині 1990-х вони випереджали свій час. Але ця мета загнала багато архітектури Яви. Однією з його головних торгових точок було «писати один раз, бігати куди завгодно», тож іншими словами ви могли написати один фрагмент коду, тоді його можна було скласти для запуску на будь-якому пристрої.

За іронією долі, Java не набула популярності з такої причини; натомість його творці змогли скористатися чимось іншим, що з’явилося у середині 1990-х: всесвітня павутина. У Java була особливість, де можна писати речі, які називаються апплетами, невеликі програми, які можна запускати всередині веб-браузера, і оскільки Інтернет набирав популярності, Java їхала на цій хвилі і стала надзвичайно успішною та популярною. Так багато веб-додатків було написано на Java, навіть якщо це не було на увазі творців, коли вони вперше задумали мову.

На творців Java сильно вплинули існуючі мови програмування, такі як C і C ++, з якими Java поділяє безліч синтаксичних подібностей. Вони використовували ці мови як приклад того, як не робити справ, тому існували певні особливості, які творці Java явно вирішили не включати, оскільки вони були клопітними для програмістів, які використовують C та C ++.

Java є мовою статичного типу, тоді як Ruby, Python та JavaScript є динамічно набраними мовами. Люди, як правило, пристрасно ставляться до різниці між мовами, які набираються статично та динамічно.

Динамічно набраний: Якщо у вас виникли програми програмування, ви ознайомитесь із ідеєю змінної. У динамічно набраній мові, наприклад, Ruby чи

JavaScript, ви можете оголосити змінну, не кажучи про те, який тип даних ви хочете вставити у цю змінну. Змінна динамічна; це може бути що завгодно - цифру чи речення тощо.

Статично типізовані: обробляючи дані на статично типових мовах, таких як Java, ми повинні оголосити, що буде містити кожна змінна. Наприклад, ця змінна буде містити числа; інша змінна буде містити текст, а інша змінна міститиме дати тощо. Це доводить, що статично набрана мова має трохи більшу структуру до неї. Існують певні помилки, які може допустити програміст, які можуть бути зафіксовані інструментами розробки, якими ми користуємося Java, перш ніж ви навіть запустите програму. Якщо ви працюєте з динамічно набраною мовою, такою як Ruby або JavaScript, ви не знаєте, що у вас є проблема, поки не запустите код, і він не вийде з ладу.

Я, як правило, віддаю перевагу мовам статичного типу. Я думаю, що добре, щоб хтось спочатку навчився статично введеній мові, оскільки є додатковий шар коду, про який потрібно подумати, і це робить змінні більш явними. На мій досвід, людям, які спочатку навчились статично введеної мови, легше вивчати динамічно набрану мову, ніж це навпаки..

3.2 Збирання проекту системою Apache Maven

Історично Nexus Repository Manager запускався як менеджер сховищ, що підтримує формат сховищ Maven, і він продовжує включати у себе чудову підтримку юзерів Apache Maven, Apache Ant / Ivy, Eclipse Aether, Gradle та інших.

У цьому розділі пояснюється конфігурація за замовчуванням, що міститься у Nexus Repository Manager Pro та Nexus Менеджер репозиторіїв Nexus, інструкції по утворенню подальших репозиторіїв Maven, а також пошуку та перегляду сховищ. Слідуйте конфігурацію інструментів для Apache Maven, Apache Ant, Gradle та інших інструментів. Приклади конфігурації використовують переваги диспетчера сховищ, що об'єднує багато репозиторіїв та відкриває їх через групу сховищ.

Формат сховища Maven

Apache Maven утворив найбільш широко використовуваний формат сховища у екосистемі розвитку Java з випуском Apache Maven 2. Він застосовується всіма нові шими версіями Apache Maven та багатьма іншими інструментами, включаючи Apach e Ivy, Gradle, sbt, Eclipse Aether і Leiningen. Більш детальну інформацію про формат можна знайти у Прикладі - Формат сховища Maven.

Формат застосовується багатьма загальнодоступними сховищами. Центральни й репозиторій є найбільшим сховищем компонентів, спрямованих на розробку на ба зі Java / JVM і за її межами, і застосовується формат сховища Maven для випуску ко мпонентів численних проектів з відкритим кодом. Він налаштований як сховище пр оксі за замовчуванням у Apache Maven і широко застосовується у інших інструментах.

На додаток до загальних функцій управління сховищами, задокументованими у
Управлінні сховищами, специфіку формату сховища Maven можна налаштувати для кожного сховища у розділі Maven 2:

Репозиторій Maven може бути налаштований так, щоб він підходив для випуск у компонентів з
використанням політики версії версії. Центральний репозиторій застосовує політику версії версії.

Безперервна розробка, як правило, виконується із версіями знімків, що підтри муються політикою версій Snapshot. Ці значення версії повинні закінчуватися на - SNAPSHOT у файлі POM. Це дозволяє повторювати завантаження, коли фактичне ч исло, що застосовується, складається із дати / часової позначки та перелічувача, а дл я видалення все ще можна використовувати рядок версії -
SNAPSHOT. Менеджер репозиторіїв та клієнтські інструменти керують файлами ме таданих, які керують цим перекладом з версії знімка до значення часової позначки.

Політика змішаної версії дозволяє підтримувати обидва підходи у одному сховищі.

Формат репозиторію Maven визначає структуру каталогів, а також умови імен ування для файлів у структурі. Apache Maven дотримується цих умов. Інші інструме нти збирання, такі як sbt та користувальницькі інструменти, історично утворювали з вичаї, які менш строго використовують структуру каталогів, порушуючи умови імен ування файлів. Компоненти, засновані на різних конвенціях цих інструментів, були опубліковані у публічних сховищах, таких як Центральний сховище. Ці інструменти покладаються на ці змінені умови.

Ви можете налаштувати макетну політику Permissive, щоб дозволити активи у сховищі, що порушують формат за замовчуванням.

Значення Strict за замовчуванням вимагає публікації та доступу до інструменті в для додержання конвенцій Apache Maven. Це найкраще налаштування, якщо ви застосовуєте Apache Maven, Eclipse Aether та інші суворо сумісні інструменти (рисунок 3.1).

```
<!-- Boot starter-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
</dependency>
```

Рисунок 3.1 – файл pom.xml блок dependency

3.3 Керування версіями системою GIT

Git - розподілена система управління версіями для відстеження змін у вихідному коді під час розроблення програмного забезпечення. Він призначений для координації роботи між програмістами, але його можна використовувати для відстеження змін у будь-якому наборі файлів. Її цілі включають швидкість, цілісність даних та підтримку розподілених, нелінійних робочих процесів.

Git був створений Лінусом Торвальдсом у 2005 році для розроблення ядра Linux, а інші розробники ядра сприяли його початковій розробці. Її нинішнім керівником з

2005 року є Хуніо Хамано. Як і в більшості інших розподілених систем управління версіями, і на відміну від більшості клієнтсько-серверних систем, кожен каталог Git на кожному комп’ютері є повноцінним сховищем із повною історією та повною здатністю відстеження версій, незалежно від доступу до мережі або центрального сервера. Git - це безкоштовне програмне забезпечення з відкритим кодом, що поширюється за умовами Загальної публічної ліцензії GNU 2.

Її розробка почалася в квітні 2005 року, після того як багато розробників ядра Linux відмовилися від доступу до BitKeeper - власної системи управління джерелами управління (SCM), яку раніше використовували для підтримки проекту. Власник авторських прав на BitKeeper Ларрі Маквей відмовився від безкоштовного використання продукту після того, як стверджував, що Ендрю Тріджллл створив SourcePuller шляхом зворотної інженерії протоколів BitKeeper. Цей же інцидент також стимулював створення іншої системи контролю версій – Mercurial, як зображено на рисунку 3.2.

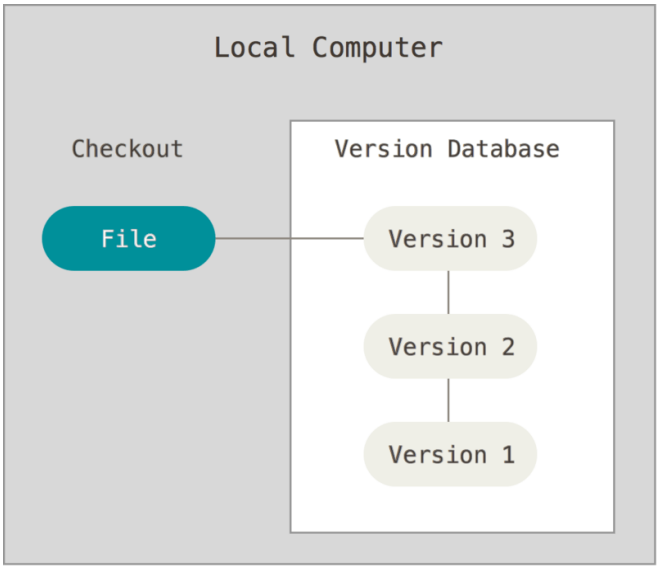

Рисунок 3.2 – Локальні версії контролю [14]

Лінус Торвальдс хотів розподілену систему, якою він міг би користуватися, як BitKeeper, але жодна з доступних безкоштовних систем не відповідала його потребам. Торвальдс наводив приклад системи управління джерелом управління, що потребує 30 секунд, щоб застосувати патч та оновити всі пов'язані з ним метадані, і зазначив,

що це не може відповідати потребам розроблення ядра Linux, де для синхронізації з колегами, що підтримують, може знадобитися 250 таких дій на один раз. Для своїх критеріїв дизайну він уточнив, що виправлення не потрібно більше трьох секунд, і додав ще три бали:

Візьміть систему паралельних версій (CVS) як приклад того, що не робити; якщо сумніваєтесь, прийміть прямо протилежне рішення.

Розробка Git розпочалася 3 квітня 2005 року. Торвальдс оголосив про проект 6 квітня; він став самостійним хостингом з 7 квітня. Перше злиття декількох гілок відбулося 18 квітня. Торвальдс досяг своїх цілей у виконанні; 29 квітня народився Git орієнтувався на патчі запису до дерева ядра Linux зі швидкістю 6,7 патча в секунду. 16 червня Git керував випуском ядра 2.6.12.

26 липня 2005 року Торвальдс передав технічне обслуговування Юніо Хамано, головному учаснику проекту. Hamano відповідав за випуск 1.0 21 грудня 2005 року та залишається виконавцем проекту(рисунок 3.3).

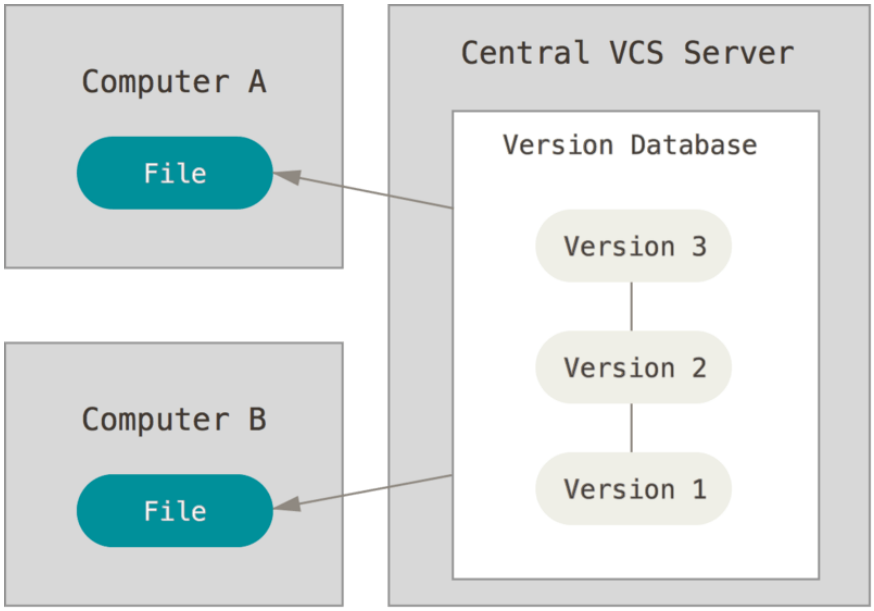

Рисунок 3.3 – Контроль версій централізований[14]

Дизайн Git був натхненний BitKeeper та Monotone. Спочатку Git був розроблений як низькорівневий двигун системи управління версіями, поверх якого інші могли писати передні частини, наприклад, Cogito або StGIT. Основний проект

Git з тих пір став повноцінною системою контролю версій, яку можна використовувати безпосередньо. Під сильним впливом BitKeeper Торвальдс свідомо уникав традиційних підходів, ведучи до унікального дизайну.

Дизайн Гіта - це синтез досвіду Торвальдса з Linux щодо підтримки великого розповсюдженого проекту розробки, а також його інтимних знань щодо продуктивності файлової системи, отриманих від цього ж проекту, та нагальної необхідності створити робочу систему в короткі терміни. Ці впливи призвели до таких варіантів реалізації .

Git підтримує швидке розгалуження та об'єднання та включає конкретні інструменти для візуалізації та навігації по нелінійній історії розвитку. Основне припущення в Git полягає в тому, що зміна буде об'єднуватися частіше, ніж написано, оскільки вона передається різним рецензентам. У Git гілки дуже легкі: гілка є лише посиланням на один коміт. Завдяки батьківським зобов'язанням може бути побудована повна галузева структура.

Як і Darcs, BitKeeper, Mercurial, Bazaar і Monotone, Git надає кожному розробнику локальну копію повної історії розробок, а зміни копіюються з одного такого сховища в інше. Ці зміни імпортуються як додаткові галузі розвитку, і їх можна об'єднати так само, як і місцево розвинену галузь.

Репозиторії можуть публікуватися через протокол передачі гіпертексту (HTTP), протокол передачі файлів (FTP) або протокол Git через звичайний сокет або захищену оболонку (ssh). Git також має емуляцію сервера CVS, що дозволяє використовувати існуючі клієнти CVS та плагіни IDE для доступу до сховищ Git. Субверсійні сховища можна використовувати безпосередньо з git-svn.

Торвальдс назвав Git дуже швидким і масштабованим, а тести на ефективність, зроблені Mozilla, показали, що він на порядок швидший, ніж деякі системи контролю версій; отримання історії версій з локально збереженого сховища може бути в сто разів швидшим, ніж його отримання з віддаленого сервера.

Історія Git зберігається таким чином, що ідентифікатор конкретної версії (фіксація в умовах Git) залежить від усієї історії розвитку, що веде до цього комітату. Після його публікації неможливо змінити старі версії, не помітивши їх. Будова схожа

на дерево Меркла, але з доданими даними про вузли та листя. (Меркуріал і Монотон також мають цю властивість.)

Git був розроблений як набір програм, написаних на C, та декількох скриптів оболонок, які надають обгортки навколо цих програм. Хоча більшість цих сценаріїв з тих пір були переписані на C для швидкості та портативності, дизайн залишається, і складно ланцюг компонентів разом.

Як частина своєї розроблення інструментарію, Git має чітко визначену модель неповного злиття, і у неї є кілька алгоритмів її завершення, що завершилось тим, що повідомляють юзереві, що він не в змозі завершити злиття автоматично і що вручну потрібно редагувати.

Відмова від операцій або резервування змін залишать у базі даних марні об'єкти, що звисають. Це, як правило, невелика частка постійно зростаючої історії шуканих предметів. Git автоматично здійснюватиме збір сміття, коли в сховищі буде створено достатньо пухких предметів. Збір сміття можна назвати явно за допомогою git gc --prune.

Git зберігає кожен новостворений об'єкт як окремий файл. Хоча індивідуально стискається, це займає багато місця і є неефективним. Це вирішується використанням пакетів, які зберігають велику кількість об'єктів, дельта-стислих між собою, в одному файлі (або мережевому байтовому потоці), який називається packfile. Пакети стискаються за допомогою евристики про те, що файли з однаковою назвою, ймовірно, схожі, але від їх коректності не залежать. Для кожного пакету файлів створюється відповідний індексний файл, який повідомляє про зміщення кожного об'єкта в пакетному файлі. Новостворені об’єкти (із нещодавно доданою історією) як і раніше зберігаються як окремі об'єкти, і для підтримання ефективності простору потрібно періодичне переупакування. Процес упаковки сховища може бути дуже обчислювально затратним. Дозволяючи об’єктам існувати у сховищі у вільному, але швидко сформованому форматі, Git дозволяє відкладати дорогу операцію з пакетом на пізніше, коли час має менше значення, наприклад, кінець робочого дня. Періодична переупаковка Git проводиться автоматично, але також можливе ручне перепакування за допомогою команди git gc. Для цілісності даних пакетний файл та

його індекс мають контрольну суму SHA-1 всередині, а ім'я файлу packfile також містить контрольну суму SHA-1. Щоб перевірити цілісність сховища, запустіть команду git fsck.

Ще одна властивість Git полягає в тому, що він робить знімки дерев файлів файлів. Найдавніші системи для відстеження версій вихідного коду, Система управління вихідним кодом (SCCS) та Система контролю версій (RCS), працювали над окремими файлами та підкреслювали економію місця, яка повинна бути отримана від перемежованих дельт (SCCS) або дельта кодування (RCS) (переважно подібні) версії. Пізніше системи контролю ревізії підтримували це поняття файлу, що має ідентичність у кількох редакціях проекту. Однак Торвальдс відкинув цю концепцію. Отже, Git не записує явно відношення відновлення файлів на будь-якому рівні нижче дерева вихідного коду.

Трохи дорожче вивчити історію змін одного файлу, ніж весь проект. Щоб отримати історію змін, що впливають на певний файл, Git повинен пройти глобальну історію, а потім визначити, чи кожна зміна модифікувала цей файл. Цей метод вивчення історії, однак, дозволяє Git з однаковою ефективністю виробляти єдину історію, що показує зміни довільного набору файлів. Наприклад, підкаталог дерева-джерела плюс пов'язаний з ним файл загального заголовка - дуже поширений випадок.

Перейменами обробляються неявно, а не явно. Поширена скарга на CVS полягає в тому, що він використовує ім’я файлу для ідентифікації його історії перегляду, тому переміщення або перейменування файлу неможливе, не перериваючи його історію чи перейменуючи історію і тим самим роблячи історію неточною. Більшість систем контролю після ревізії після CVS вирішують це, надаючи файлу унікальне довгоживуче ім'я (аналогічне числу inode), яке переживе перейменування. Git не записує такого ідентифікатора, і це заявляється як перевага. Файли вихідного коду іноді розбиваються або об'єднуються, або просто перейменовуються, і записавши це як просте перейменування, заморозили б неточний опис того, що сталося в (незмінній) історії. Git вирішує проблему, виявляючи перейменування під час перегляду історії знімків, а не записуючи її під час створення знімка. (Коротко

кажучи, з огляду на файл у версії N, його однойменним предком є те саме ім'я в редакції N - 1. Однак, коли в редакції N - 1 не існує файлу з назвою, який називається однотипним іменем, Git шукає файл, який існував лише в редакції N - 1 і дуже схожий на новий файл.) Однак для цього потрібно вимагати більш інтенсивної роботи процесора щоразу, коли історія переглядається, і є кілька варіантів коригування евристики. Цей механізм не завжди працює; іноді файл, який перейменований із змінами в той самий коміт, читається як видалення старого файлу та створення нового файлу. Розробники можуть подолати це обмеження, виконавши перейменування та зміни окремо.

Git реалізує кілька стратегій злиття; під час злиття може бути обрана стратегія, що не використовується за замовчуванням (рисунок 3.4).

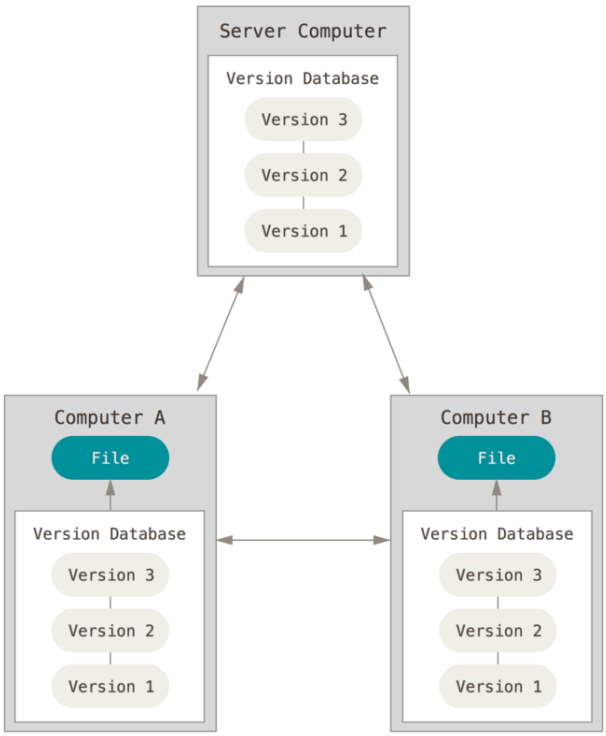

Рисунок 3.4 –Контроль версій розподілений [14].

Коли існує більше одного спільного предка, який може бути використаний для тристороннього злиття, він створює об'єднане дерево спільних предків і використовує це як опорне дерево для тристороннього злиття. Повідомлялося, що це призводить до меншої кількості конфліктів злиття, не викликаючи помилок злиття за допомогою

тестів, зроблених на попередніх комісіях злиття, взятих з історії розроблення ядра Linux. Крім того, це дозволяє виявити та обробити злиття, що включають перейменування.

Git не забезпечує механізми контролю доступу, але був розроблений для роботи з іншими інструментами, що спеціалізуються на контролі доступу.

17 грудня 2014 року було знайдено подвиг, що стосується версій Windows та macOS для клієнта Git. Зловмисник може виконувати довільне виконання коду на цільовому комп'ютері із встановленим Git, створивши шкідливе дерево Git (каталог) з назвою .git (каталог у сховищах Git, який зберігає всі дані сховища) в іншому випадку (наприклад, .GIT або .Git, необхідний тому, що Git не дозволяє створювати всю малу версію .git вручну) зі шкідливими файлами у підкаталозі .git / hooks (папці із виконуваними файлами, які запускає Git) у сховищі, яке створив зловмисник або у сховищі, яке може змінити зловмисник. Якщо юзер Windows або Mac витягує (завантажує) версію сховища зі шкідливим каталогом, а потім переходить до цього каталогу, .git каталог буде перезаписаний (через нечутливі до регістру риси файлових систем Windows та Mac) та можуть запускатися шкідливі виконувані файли в .git / hooks, що призводить до виконання команд зловмисника. Зловмисник також може змінити файл конфігурації .git / config, який дозволяє зловмиснику створювати шкідливі псевдоніми Git (псевдоніми для Git-команд або зовнішніх команд) або змінювати існуючі псевдоніми для виконання шкідливих команд під час запуску. Уразливість була виправлена у версії 2.2.1 Git, випущеній 17 грудня 2014 року та оголошена наступного дня.

Git версія 2.6.1, випущена 29 вересня 2015 року, містила виправлення для вразливості безпеки (CVE-2015-7545), що дозволяло виконувати довільне виконання коду. Уразливість була корисною, якщо зловмисник міг переконати жертву в клонуванні певної URL-адреси, оскільки довільні команди були вбудовані в саму URL-адресу. Зловмисник може скористатися експлуатуванням через атаку "людина-посеред", якщо з'єднання було незашифровано, оскільки вони могли перенаправити юзера на обрану ним URL-адресу. Рекурсивні клони також були вразливими, оскільки дозволили контролеру сховища вказати довільні URL-адреси через файл gitmodules.

Git використовує хеші SHA-1 внутрішньо. Лінус Торвальдс відповів, що хеш в основному захищав від випадкової корупції, а безпека, яку дає криптографічно захищений хеш, була лише випадковим побічним ефектом, а основна безпека підписувалася в іншому місці.

3.4 Розробка модулів з використанням фреймворку Spring Boot

Spring - це структура з відкритим кодом для розробки додатків та управління інвестиційним контейнером для платформи Java.

Перша версія була написана Родом Джонсоном, який випустив її, опублікувавши книгу "Експерт Один на один" Дизайн та розробка J2EE (Wrox Press, жовтень 2002). Рамка була спочатку запущена за ліцензією Apache 2.0 у червні 2003 року. Першим основним випуском була версія 1.0, яка з'явилася в березні 2004 року, а за нею відбулися інші віхи у вересні 2004 року та березні 2005 року. Версія 1.2.6 Spring Framework виграв нагороди Jolt Awards та Jax Innovation Awards в 2006 році. Spring Framework 2.0 був запущений у 2006 році, версія 2.5 у листопаді 2007 року, весна 3.0 у грудні 2009 року та весна 3.1 через два роки (рисунок 3.5).

```
@Slf4j
@Configuration
public class TwitterConfig {
    @Value("${spring.social.twitter.app-id}")
    private String twitterId;
    @Value("${spring.social.twitter.app-secret}")
    private String twitterSecret;
    @Value("${spring.social.twitter.token-id}")
    private String twitterTokenId;
    @Value("${spring.social.twitter.token-secret}")
    private String twitterTokenSecret;
    @Bean
    public StreamingOperations streamingOperations() { return twitter().streamingOperations(); }

    @Bean
    public UserOperations userOperations() { return twitter().userOperations(); }

    @Bean
    public Twitter twitter(){
        return new TwitterTemplate(twitterId, twitterSecret, twitterTokenId, twitterTokenSecret);
    }
}
```

Рисунок 3.5 – Twitter сервіс конфігурація

Хоча основні особливості Spring Framework можна використовувати в будь-якому додатку, розробленому на Java, існує декілька розширень для побудови веб-додатків на платформі Java EE. Хоча вона не нав'язує якоїсь конкретної моделі програмування, ця структура набула популярності у спільноті, оскільки її вважають альтернативною, заміною і навіть доповненням до моделі EJB (Enterprise JavaBean).

Spring Framework складається з декількох модулів, які надають цілий спектр послуг.

Контейнер інвестицій управління: він дозволяє конфігурувати компоненти додатків і адмініструвати цикл життєдіяльності об'єктів Java, здійснюється в основному за допомогою введення залежностей.

Програмно-орієнтоване програмування: дозволяє реалізувати поперечні підпрограми.

Доступ до даних: RDBMS працює на платформі java, використовуючи Зв'язок бази даних Java та інструменти відображення, пов’язані з базами даних NoSQL.

Управління транзакціями: уніфікує різні API управління та координує транзакції для об'єктів Java.

Модель перегляду контролера: Рамка на основі HTTP та сервлетів, яка надає інструменти для розширення та налаштування веб-додатків та веб-служб REST.

Рамка віддаленого доступу: Дозволяє імпортувати та експортувати об’єкти Java у стилі RPC через мережі, що підтримують протоколи, засновані на RMI, CORBA та HTTP, включаючи веб-сервіси (SOAP).

Конвенція про конфігурацію: модуль Spring Roo пропонує швидке рішення для розробки програм на основі Spring Framework, привілейовану простоту без втрати гнучкості.

Пакетна обробка: основа для об'ємної обробки, яка містить журнал / побудову графіків, обробку транзакцій, статистику обробки завдань, перезапуск завдання та функції управління ресурсами.

Аутентифікація та авторизація: налаштовані процеси захисту, які підтримують цілий ряд стандартів, протоколів, інструментів та практик через підпроект Spring Security (раніше Acegi) (рисунок 3.6).

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
            .csrf().disable()
            .authorizeRequests()
            .antMatchers("/", "/signin/**", "/callback").permitAll()
            .anyRequest().authenticated();
    http
            .formLogin()
            .loginPage("/")
            .permitAll();
    http
            .logout()
            .deleteCookies(COOKIE_J_SESSION_ID)
            .permitAll()
            .and().apply(new SpringSocialConfigurer());
}
```

Рисунок 3.6 – Рівень безпеки та його налаштування

Віддалене адміністрування: конфігурація видимості та управління об'єктами Java для локальної або віддаленої конфігурації через JMX (рисунок 3.7).

```
@Repository
public interface UserRepository extends MongoRepository<User, String> {
    User findUserByUsername(String username);
}
```

Рисунок 3.7 – Репозиторій юзера

Повідомлення: Налаштований реєстр об'єктів прийому повідомлень, для прозорого споживання з СМС, покращення надсилання повідомлень на стандартні API-адреси JMS.

Тестування: Підтримка класу для розробки та інтеграції тестових блоків. Приклад тестового сценарію зображено на рисунку 3.8.

```
@RunWith(SpringRunner.class)
public class OrganizationServiceTest {
    @Mock
    private OrganizationRepository organizationRepository;
    @InjectMocks
    private OrganizationServiceImpl organizationService;
    private Organization organization;

    @Test
    public void getCurrentOrganization() {
        when(organizationRepository.findAll()).thenReturn(Collections.singletonList(organization));
        Optional<Organization> currentOrganization = organizationService.getCurrentOrganization();
        assertEquals(of(organization), currentOrganization);
    }
```

Рисунок 3.8 – Тест оцінки якості аналізу

Серцем Spring Framework є його інвестиційний контейнер управління (IoC). Його завдання - інстанціювати, ініціалізувати та підключати об’єкти програми, на додаток до надання ряду додаткових функцій, доступних навесні протягом усього життя об'єктів (рисунок 3.9).

```
@GetMapping
public ResponseEntity<String> verifyWebhook(String mode,
                                            String verifyToken,
                                            String challenge) {
    try {
        this.messenger.verifyWebhook(mode, verifyToken);
        log.info("Webhook verification done");
        return ResponseEntity.status(HttpStatus.OK).build();
    } catch (MessengerVerificationException e) {
        log.warn("Webhook verification failed: {}", e.getMessage());
        return ResponseEntity.status(HttpStatus.FORBIDDEN).body(e.getMessage());
    }
}
```

Рисунок 3.9 – Контроллер чат бота

Об'єкти, створені та керовані контейнером, називаються керованими об'єктами або квасолями. Ці об'єкти мають тип POJO. Для виконання свого завдання контейнеру потрібна інформація, яка вказує, як інстанціювати і з'єднувати квасоля один з одним. Ця інформація називається метаданими конфігурації. Існують різні способи надання цієї інформації: на основі XML, на основі анотацій або на об'єктах Java (починаючи з Spring 3.0). Контейнер не залежить від формату метаданих конфігурації. Користувач

може використовувати потрібний формат і навіть змішувати їх в одній програмі (рисунку 3.10).

```
@Override
@Transactional(propagation = Propagation.REQUIRED)
public void createUser(String facebookUserId) throws MessengerApiException, MessengerIOException {
    UserProfile userProfile = messenger.queryUserProfile(facebookUserId);

    userService.create(User.builder()
            .facebookId(facebookUserId)
            .username(facebookUserId)
            .userRoles(Stream.of(ROLE_USER, ROLE_MESSENGER).collect(Collectors.toSet()))
            .firstName(userProfile.firstName())
            .lastName(userProfile.lastName())
            .locale(userProfile.locale())
            .build());
}
```

Рисунок 3.10 – Керування транзакціями сервісу

Об'єкти можуть бути отримані шляхом пошуку залежності або шляхом введення залежності пошук залежностей - це модель, де об’єкт із конкретним іменем або певного типу запитується від об'єкта контейнера. Введення залежності - це модель, в якій контейнер передає об'єкти за назвою іншим об'єктам, або через конструкторські методи, властивості або фабричні методи.

У багатьох випадках, коли ви використовуєте інші частини Spring Framework, вам не потрібно використовувати контейнер, хоча його використання, ймовірно, дозволяє полегшити налаштування та налаштування програми. Spring Container надає вам послідовний механізм налаштування програм і інтегрується майже з усіма середовищами Java, від невеликих додатків до великих бізнес-додатків.

Контейнер можна перетворити в контейнер EJB 3.0 частково за допомогою проекту Pitchfork. Деякі критикують Spring Framework за те, що він не відповідає стандартам. Однак SpringSource не вважає відповідність EJB 3 важливою метою, і стверджує, що Spring Framework та контейнер дозволяють використовувати більш потужні моделі програмування. Не створюйте об'єкт , але опишіть спосіб їх створення, визначивши його у файлі конфігурації Spring. Ви не дзвоните до служб та компонентів, але ви кажете, які служби та компоненти слід викликати, визначаючи їх

у файлах конфігурації Spring. Це робить код легким у обслуговуванні та простішим тестуванням за допомогою залежної ін'єкції (IoC) (рисунок 3.11).

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Рисунок 3.11 – Стартова точка розгортання системи

3.5 Брокер повідомлень Kafka

Apache Kafka - це платформа програмного забезпечення з відкритим вихідним кодом, розроблена LinkedIn та передана Фонду програмного забезпечення Apache, написаного на Scala та Java. Проект спрямований на створення єдиної, високопродуктивної, низької затримки платформи для обробки каналів даних у режимі реального часу. Kafka може підключатися до зовнішніх систем (для імпорту / експорту даних) через Kafka Connect та надає Kafka Streams, бібліотеку обробки потоків Java.

Kafka використовує двійковий протокол на основі TCP, який оптимізований для ефективності та покладається на абстракцію "набору повідомлень", яка, природно, групує повідомлення, щоб зменшити накладні витрати в мережі. Це "призводить до більших мережевих пакетів, більших послідовних дискових операцій, безперервних блоків пам'яті , що дозволяє Кафці перетворити бурхливий потік випадкових повідомлень, що записуються в лінійні записи".

Apache Kafka спочатку був розроблений LinkedIn, а згодом був відкритий на початку 2011 року. Закінчення інкубатора Apache відбулося 23 жовтня 2012 року. За повідомленням Quora від 2014 року, Kreps вибрала назву програмного забезпечення на ім'я автора Франца Кафки, оскільки це "система, оптимізована для написання", і йому сподобалася робота Кафки.

Apache Kafka заснований на журналі фіксації, і він дозволяє користувачам підписатися на нього та публікувати дані в будь-якій кількості систем або додатків у режимі реального часу. Приклади застосувань включають управління сполученням пасажирів та водіїв в Uber, надання аналітики в реальному часі та прогнозного обслуговування для розумного будинку British Gas та надання численних послуг у режимі реального часу по всій LinkedIn.

Kafka зберігає ключові повідомлення, що надходять від довільно багатьох процесів, званих виробниками. Дані можна розділити на різні "розділи" в межах різних "тем". У розділі повідомлення суворо впорядковуються за їх зміщенням (положення повідомлення у розділі) та індексуються та зберігаються разом із позначкою часу. Інші процеси, які називаються "споживачами", можуть читати повідомлення з розділів. Для потокової обробки Kafka пропонує Streams API, який дозволяє записувати програми Java, які споживають дані з Kafka, і записувати результати назад у Kafka. Apache Kafka також працює із системами обробки зовнішніх потоків, такими як Apache Apex, Apache Flink, Apache Spark та Apache Storm.

Kafka працює на кластері одного або декількох серверів (званих брокерами), а розділи всіх тем розподіляються по вузлах кластера. Крім того, розділи реплікуються на кілька посередників. Ця архітектура дозволяє Kafka доставляти масивні потоки повідомлень у відмовостійкому режимі і дозволила їй замінити деякі звичайні системи обміну повідомленнями, такі як Java Message Service (JMS), Advanced Message Queuing Protocol (AMQP) тощо. З 0.11. 0.0 випуску, Kafka пропонує транзакційні записи, які забезпечують обробку потоку точно-раз за допомогою Streams API (рисунок 3.12).

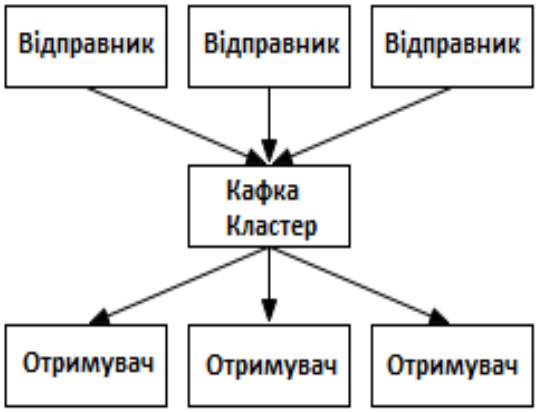

Рисунок 3.12 – Централізоване оброблення повідомлень брокером [17]

Kafka підтримує два типи тем: регулярні та ущільнені. Регулярні теми можна налаштувати за допомогою часу утримування або обмеження пробілу. Якщо є записи, старші зазначеного часу збереження, або якщо обмежений простір перевищено для розділу, Kafka дозволено видаляти старі дані до вільного місця зберігання. За замовчуванням теми налаштовуються із часом зберігання 7 днів, але також можна зберігати дані на невизначений час. Для компактних тем записи не закінчуються на основі меж часу або простору. Натомість Кафка розглядає пізніші повідомлення як оновлення до старих повідомлень тим самим ключем і гарантує ніколи не видаляти останнє повідомлення за ключем. Користувачі можуть видаляти повідомлення повністю, написавши так зване надгробне повідомлення з нульовим значенням для конкретного ключа. На рисунку 3.13 зображено поділ топіка на партішини.

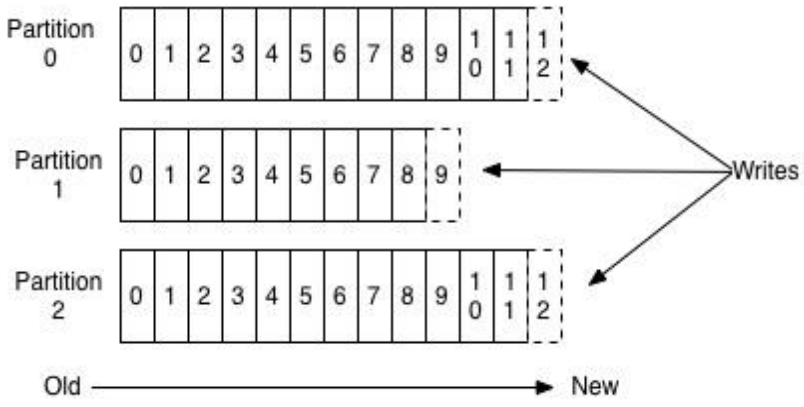

Рисунок 3.13 – Структура топіка [17]

Consumer API - дозволяє додатку підписатися на теми та обробляти потоки записів.

API Connector - виконує API для багаторазового використання для виробників та споживачів, які можуть пов'язувати теми з існуючими програмами.

API потоків - цей API перетворює вхідні потоки на вихід і видає результат.

API споживачів та виробників будуються на основі протоколу обміну повідомленнями Kafka і пропонують референтну реалізацію для клієнтів Kafka і споживачів на Java. Базовий протокол обміну повідомленнями - це бінарний протокол, який розробники можуть використовувати для написання власних споживачів або клієнтів-виробників будь-якою мовою програмування. Це відкриває Kafka з екосистеми Java Virtual Machine (JVM). Список доступних клієнтів, які не є Java, зберігається у вікі Apache Kafka.

Kafka Connect (або Connect API) є основою для імпорту / експорту даних з / в інші системи. Він був доданий у випуску Kafka 0.9.0.0 і використовує API виробника та споживача внутрішньо. Сама рамка Connect виконує так звані "роз'єми", які реалізують фактичну логіку зчитування / запису даних з інших систем. API API визначає інтерфейс програмування, який повинен бути реалізований для створення спеціального з'єднувача. Вже є багато відкритих і комерційних роз'ємів для популярних систем передачі даних. Однак сам Apache Kafka не включає в себе готові для виробництва роз'єми.

3.6 Оброблення потоків з використанням Kafka Stream

Kafka Streams (або API Streams) - це бібліотека потокової обробки, написана на Java. Він був доданий у випуску Kafka 0.10.0.0. Бібліотека дозволяє розробити потужні додатки для обробки потоку, які є масштабованими, еластичними та повністю відмовними. Основний API - це потіково-обробна мова, орієнтована на домен (DSL), яка пропонує операторів високого рівня, таких як фільтр, карта, групування, відкривання вікон, агрегація, приєднання та поняття таблиць. Крім того,

API Processor може використовуватися для реалізації спеціальних операторів для підходу до розробки на більш низькому рівні. API DSL та процесора теж може бути змішаним. Для стаціонарної обробки потоків Kafka Streams використовує RocksDB для підтримки стану локального оператора. Оскільки RocksDB може записувати на диск, підтримуваний стан може бути більшим, ніж наявна основна пам'ять. Для відмовостійкості всі оновлення місцевих державних магазинів також записуються в тему в кластері Kafka. Це дозволяє відтворити стан, читаючи ці теми та подавати всі дані в RocksDB.

До версії 0.9.x, брокери Kafka підтримують сумісність лише зі старими клієнтами. Оскільки Kafka 0.10.0.0, брокери також підтримують сумісність з новими клієнтами. Якщо новий клієнт підключається до старшого брокера, він може використовувати лише ті функції, які підтримує брокер. Для API Streams повна сумісність починається з версії 0.10.1.0: програма 0.10.1.0 Kafka Streams не сумісна з брокерами 0.10.0 або старішими.

Моніторинг ефективності роботи в кінці вимагає відстеження показників від посередників, споживачів та виробників, крім моніторингу ZooKeeper, який Kafka використовує для координації між споживачами. В даний час існує декілька платформ моніторингу для відстеження продуктивності Kafka, або з відкритим кодом, наприклад, Burrow, LinkedIn, або платними, як Datadog. На додаток до цих платформ, збір даних Kafka також може здійснюватися за допомогою інструментів, які зазвичай входять до Java, включаючи JConsole.

Топіки можуть розростатися, тому великі топіки поділяються на більш дрібні секції партішини для поліпшення продуктивності і масштабованості.

3.7 Розробка моделі бази даних MongoDB

MongoDB - це міжплатформна програма, орієнтована на документи. MongoDB, класифікований як програма баз даних NoSQL, використовує JSON-подібні документи зі схемою. MongoDB розроблений компанією MongoDB Inc. та ліцензується під публічною ліцензією сервера (SSPL).

Компанія з програмним забезпеченням 10gen почала розробляти MongoDB в 2007 році як складову запланованої платформи як сервісного продукту. У 2009 році компанія перейшла до моделі розвитку з відкритим кодом, компанія пропонує комерційну підтримку та інші послуги. У 2013 році 10gen змінив назву на MongoDB Inc.

20 жовтня 2017 року MongoDB стала публічно проданою компанією, котирується в NASDAQ як MDB з ціною IPO в розмірі $ 24 за акцію.

30 жовтня 2019 року MongoDB об'єднався з хмарою Alibaba (NYSE: BABA), яка запропонує своїм клієнтам рішення MongoDB як послуги. Клієнти можуть використовувати керовану пропозицію з глобальних центрів обробки даних BABA.

Спеціальні запити MongoDB підтримує пошук у полі, діапазоні та регулярному пошуку виразів. Запити можуть повертати конкретні поля документів, а також включати визначені користувачем функції JavaScript. Запити також можна налаштувати для повернення випадкової вибірки результатів заданого розміру.

Поля в документі MongoDB можна індексувати первинними та вторинними індексами.

MongoDB забезпечує високу доступність наборів реплік. Набір реплік складається з двох або більше копій даних. Кожен член набору реплік може виконувати роль первинної чи вторинної репліки в будь-який час. Усі записи і зчитування за замовчуванням виконуються в основній репліку. Вторинні репліки підтримують копію даних первинного за допомогою вбудованої реплікації. Коли первинна репліка виходить з ладу, набір реплік автоматично проводить виборчий процес, щоб визначити, яка вторинна повинна стати первинною. Вторинні додатки можуть додатково обслуговувати операції зчитування, але ці дані зрештою узгоджуються за замовчуванням.

MongoDB масштабує горизонтально за допомогою заточування. Користувач вибирає клаптиковий ключ, який визначає спосіб розподілу даних у колекції. Дані поділяються на діапазони (засновані на клавіші осколки) і розподіляються по декількох фрагментах. (Осколок - це майстер з однією чи кількома репліками.). Крім

того, ключ осколка може бути хешированим, щоб відобразити фрагмент - що дозволяє рівномірний розподіл даних.

MongoDB може працювати на декількох серверах, врівноважуючи завантаження або дублюючи дані, щоб підтримувати роботу системи та працювати у разі відмови обладнання.

MongoDB може використовуватися як файлова система під назвою GridFS, з функціями балансування навантаження та реплікації даних на декількох машинах для зберігання файлів. Ця функція, яка називається сітковою файловою системою, включена в драйвери MongoDB. MongoDB відкриває розробникам функції для обробки файлів та вмісту. Доступ до GridFS можна отримати за допомогою утиліти mongofiles або плагінів для Nginx та lighttpd. GridFS ділить файл на частини або фрагменти і зберігає кожен із цих фрагментів як окремий документ.

MongoDB пропонує три способи виконання агрегації: конвеєрний конвеєр, функцію зменшення карти та одноцільові методи агрегації.

Зменшення карт може використовуватися для пакетної обробки даних та операцій агрегації. Але згідно з документацією MongoDB, трубопровід агрегації забезпечує кращу ефективність для більшості операцій агрегації.

Рамка агрегації дозволяє користувачам отримувати такі результати, для яких використовується пункт SQL GROUP BY. Оператори агрегації можуть бути з'єднані разом, щоб утворювати трубопровід - аналог труб Unix. Рамка агрегації включає оператор пошуку $, який може приєднувати документи з декількох колекцій, а також статистичні оператори, такі як стандартне відхилення.

JavaScript може використовуватися в запитах, функціях агрегації (таких як MapReduce) і надсилатися безпосередньо до бази даних, яку потрібно виконати.

MongoDB підтримує колекції фіксованого розміру, які називаються обмеженими колекціями. Цей тип колекції підтримує порядок вставки та, як тільки буде досягнуто заданого розміру, веде себе як кругова черга.

Підтримка транзакцій ACID з кількома документами була додана до MongoDB із загальною доступністю версії 4.0 у червні 2018 року. MongoDB Community Edition безкоштовна і доступна для Windows, Linux та OS X. MongoDB Enterprise Server -

комерційне видання MongoDB, доступне у складі розширеної підписки на MongoDB Enterprise.

MongoDB також доступний як повністю керований сервіс на вимогу. MongoDB Atlas працює на AWS, Microsoft Azure та Google Cloud Platform.

Серверна платформа MongoDB Stitch прискорює розробку додатків за допомогою простого, безпечного доступу до даних та послуг від клієнта - швидше отримуйте ваші програми на ринку, скорочуючи операційні витрати та зусилля.

Доступність мови програмування MongoDB має офіційні драйвери для основних мов програмування та середовищ розробки. Існує також велика кількість неофіційних або підтримуваних спільнотою драйверів для інших мов та систем програмування.

MongoDB Stitch забезпечує безсерверний доступ до MongoDB та інших послуг. Клієнтські бібліотеки доступні для JavaScript, iOS та Android.

Основним інтерфейсом до бази даних була оболонка монго. Оскільки MongoDB 3.2, MongoDB Compass представлений як рідний графічний інтерфейс. Є продукти та сторонні проекти, які пропонують користувальницькі інтерфейси для адміністрування та перегляду даних.

Станом на жовтень 2018 року MongoDB випускається за ліцензією SSPL, ліцензією, розробленою проектом. Він замінює загальну публічну ліцензію GNU Affero і майже ідентичний ліцензії GNU General Public License 3, але вимагає, щоб ті, які роблять програмне забезпечення загальнодоступним як частина "послуги", повинні зробити весь вихідний код служби доступним за цією ліцензією. SSPL був поданий на сертифікацію до Ініціативи з відкритим кодом, але пізніше був відкликаний. Мовні драйвери доступні за ліцензією Apache. Крім того, MongoDB Inc. пропонує власні ліцензії на MongoDB. Останні версії, ліцензовані як AGPL версії 3.

MongoDB було вимкнено з дистрибутивів Debian, Fedora та Red Hat Enterprise Linux через зміну ліцензії. Fedora визначив, що версія 1 SSPL не є ліцензією на вільне програмне забезпечення, оскільки "навмисно створена для того, щоб бути агресивно дискримінаційною" щодо комерційних користувачів. MongoDB Enterprise Server MongoDB пропонує як Enterprise, так і спільноту своєї потужної бази розподілених документів. MongoDB Enterprise доступний у складі розширеної підписки на

MongoDB Enterprise, яка містить найбільш повну підтримку та найкращий угода про надання послуг за умовами обслуговування, коли ви запускаєте MongoDB у власній інфраструктурі. MongoDB Enterprise Advanced також надає вам комплексне операційне оснащення, вдосконалену аналітику та візуалізацію даних, інтеграцію платформ та сертифікацію, а також навчання на замовлення для ваших команд.

Через конфігурацію безпеки MongoDB за замовчуванням, що дозволяє кожному мати повний доступ до бази даних, було викрадено дані з десятків тисяч установок MongoDB. Крім того, багато серверів MongoDB було проведено за викуп.

Починаючи з версії MongoDB 2.6, двійкові файли з офіційних пакетів MongoDB RPM та DEB прив'язуються до localhost за замовчуванням. З MongoDB 3.6 ця поведінка за замовчуванням поширювалася на всі пакети MongoDB на всіх платформах. В результаті всі мережеві з'єднання з базою даних будуть відхилені, якщо явно не налаштований адміністратором.

До версії 3.3.11, MongoDB не міг проводити сортування на основі порівняння і обмежувався байт-порівнянням через memcmp, що не забезпечило б правильне впорядкування для багатьох не англійських мов при використанні з кодуванням Unicode. Виправлення виправлено 23 серпня 2016 року.

До MongoDB 4.0 запити проти індексу не були атомними. Документи, які оновлювалися під час запуску запиту, можуть бути пропущені. Впровадження проблеми MongoDB 4.0, що зачіпає знімок, усунув це явище.

3.8 Утворення контейнерів Docker

Docker - це набір платформ як сервісних (PaaS) продуктів, які використовують віртуалізацію на рівні ОС для доставки програмного забезпечення в пакетах, званих контейнерами. Контейнери відокремлені один від одного і поєднують власне програмне забезпечення, бібліотеки та файли конфігурації; вони можуть спілкуватися між собою через чітко визначені канали. Усі контейнери управляються одним ядром операційної системи і, таким чином, більш легкі, ніж віртуальні машини.

Послуга має як безкоштовний, так і преміальний рівні. Програмне забезпечення, в якому розміщуються контейнери, називається Docker Engine. Вперше він був запущений у 2013 році та розроблений компанією Docker, Inc.

Докер, була заснована Соломоном Хайкесом та Себастьєном Палем під час групи інкубаторів запуску Y Combinator Summer 2010 і була запущена в 2011 році. Hykes розпочав проект Docker у Франції як внутрішній проект в рамках dotCloud, компанії платформи як послуга.

Докер дебютував публіці в Санта-Кларі на PyCon в 2013 році. Він був випущений як відкритий код у березні 2013 року. У той час LXC використовував як середовище виконання за замовчуванням. Через рік, випустивши версію 0.9, Docker замінив LXC на власний компонент, який був написаний мовою програмування Go.

19 вересня 2013 року Red Hat та Docker оголосили про співпрацю навколо Fedora, Red Hat Enterprise Linux (RHEL) та OpenShift. 15 жовтня 2014 р. Microsoft оголосила про інтеграцію двигуна Docker у Windows Server, а також власну підтримку ролі клієнта Docker у Windows.

У листопаді 2014 року послуги контейнерів Docker були оголошені для обчислювальної хмари Amazon Elastic (EC2). 10 листопада 2014 року Докер оголосив про партнерство зі Стратоскале. 4 грудня 2014 року IBM оголосила про стратегічне партнерство з Docker, що дозволяє Docker тісніше інтегруватися з IBM Cloud. 22 червня 2015 року Docker та кілька інших компаній оголосили, що працюють над новим стандартом для контейнерів програмного забезпечення, незалежним від постачальника та операційної системи. У квітні 2016 року Windocks, незалежний ISV, випустив порт Windows з відкритим кодом проекту Docker для Windows, що підтримує Windows Server 2012 R2 та Server 2016, з усіма версіями SQL Server 2008 і далі.

Аналіз в травні 2016 року показав такі організації як основні учасники Docker: команда Docker, Cisco, Google, Huawei, IBM, Microsoft та Red Hat. 8 червня 2016 року Microsoft оголосила, що тепер Docker тепер можна використовувати в оригінальному режимі в Windows 10.

Аналіз згаданих профілів LinkedIn у січні 2017 року показав, що присутність Докера зросла на 160% у 2016 році.

6 травня 2019 року Microsoft оголосила про другу версію підсистеми Windows для Linux (WSL). Docker, Inc. оголосила, що почала працювати над версією Docker для Windows, яка працює на WSL 2.

Docker може упакувати програму та її залежності у віртуальний контейнер, який може працювати на будь-якому сервері Linux. Це допомагає забезпечити гнучкість та портативність, що дозволяє запускати програму в різних місцях, незалежно від локальних даних, у відкритій хмарі чи в приватній хмарі. Docker використовує функції виділення ресурсів ядра Linux (наприклад, групи груп та простори імен ядра) та файлову систему, що підтримує об'єднання (наприклад, OverlayFS), щоб контейнери могли працювати в одному екземплярі Linux, уникаючи накладних витрат на запуск та підтримку віртуальних машин . Оскільки контейнери Docker мають невелику вагу, один сервер або віртуальна машина можуть запускати кілька контейнерів одночасно. Аналіз 2018 року встановив, що типовий випадок використання Docker передбачає запуск восьми контейнерів на хоста, але що чверть проаналізованих організацій працює 18 або більше на одного хоста.

Підтримка ядра Linux для просторів імен здебільшого ізолює уявлення програми про операційне середовище, включаючи дерева процесів, мережу, ідентифікатори користувачів та змонтовані файлові системи, тоді як групи ядер забезпечують обмеження ресурсів для пам'яті та процесора. Починаючи з версії 0.9, Docker включає власний компонент (званий "libcontainer") для прямого використання засобів віртуалізації, що надаються ядром Linux, на додаток до використання абстрагованих інтерфейсів віртуалізації через libvirt, LXC та systemd-nspawn.

Docker реалізує API високого рівня, щоб забезпечити легкі контейнери, які запускають процеси ізольовано. Програмне забезпечення Docker як послуга пропонує три компоненти.

Програмне забезпечення: Демон Docker, який називається dockerd, - це постійний процес, який управляє контейнерами Docker і обробляє контейнерні об'єкти. Демон слухає запити, надіслані через API Docker Engine. Клієнтська

програма Docker, що називається docker, забезпечує інтерфейс командного рядка, що дозволяє користувачам взаємодіяти з демонами Docker.

Об'єкти: Докерські об'єкти - це різні об'єкти, які використовуються для збирання програми в Докер. Основні класи об’єктів Docker - зображення, контейнери та послуги.

Контейнер Docker - це стандартизоване, капсульоване середовище, яке запускає програми. Керує контейнером за допомогою API Docker або CLI.

Зображення Докера - це шаблон лише для читання, який використовується для створення контейнерів. Зображення використовуються для зберігання та доставки програм.

Служба Docker дозволяє масштабувати контейнери через кілька демонів Docker. Результат відомий як рій, набір демонів, що співпрацюють, що спілкуються через Docker API.

Реєстри: Реєстр Docker - це сховище для зображень Docker. Клієнти Docker підключаються до реєстрів, щоб завантажити ("потягнути") зображення для використання або завантажити ("натиснути") створені ними зображення. Реєстри можуть бути державними або приватними. Два основні державні реєстри - Docker Hub та Docker Cloud. Docker Hub - це реєстр за замовчуванням, де Docker шукає зображення. Реєстри докерів також дозволяють створювати сповіщення на основі подій.

Docker Compose - це інструмент для визначення та запуску багатоконтейнерних програм Docker. Він використовує файли YAML для налаштування служб програми та виконує процес створення та запуску всіх контейнерів за допомогою однієї команди. Утиліта CLI для докер-компонування дозволяє користувачам запускати команди на декількох контейнерах одночасно, наприклад, створення зображень, масштабування контейнерів, запуску контейнерів, які були зупинені тощо. Команди, пов’язані з маніпулюванням зображеннями або інтерактивними параметрами користувачів, у Docker Compose не мають значення, оскільки вони адресують один контейнер. Файл docker-compose.yml використовується для визначення послуг програми та включає різні параметри конфігурації. Наприклад, опція збірки визначає

параметри конфігурації, такі як шлях Dockerfile, команда дозволяє переосмислити команди Docker за замовчуванням тощо. Перша публічна бета-версія Docker Compose (версія 0.0.1) була випущена 21 грудня 2013 р. Перша готова до виробництва версія (1.0) була доступна 16 жовтня 2014 року.

Docker Swarm забезпечує вбудовану функцію кластеризації контейнерів Docker, що перетворює групу двигунів Docker в єдиний віртуальний движок Docker. У режимі Docker 1.12 і вище режим Swarm інтегрований з Docker Engine. Утиліта CLI роя докера дозволяє користувачам запускати контейнери Swarm, створювати маркери відкриття, перелічувати вузли в кластері та інше. Утиліта CLI-вузла докерного вузла дозволяє користувачам виконувати різні команди для управління вузлами в рої, наприклад, перелічуючи вузли в рій, оновляючи вузли та видаляючи вузли з рою. Докер керує роями, використовуючи алгоритм консенсусу на плоті. За словами Рафта, для того, щоб здійснити оновлення, більшість вузлів Swarm потребують узгодження оновлення.

3.9 Висновки до розділу 3

Отже, в даному розділі було проведено аналіз та обґрунтування вибору технічних засобів для реалізації системи та її компонентів. Дани аналіз допогає розробникам програмного продукту реалізувати поставлені задачі. Тому у наступному розділі буде представлено результати розробленого прототипу системи.

4 ТЕСТУВАННЯ ТА ОСОБЛИВОСТІ РОБОТИ СИСТЕМИ

4.1 Утворення пошукового запиту

Юзери системи утворюють та відправляють запит, указавши запит на цільові словосполучення у системі Twitter. Для спортивних фанатів, наприклад, юзери можуть ввести пошукові словосполучення пов’язані з футболом, наприклад, «premierleague», «football» та командні імена, як «Liverpool» та «Manchester City». Юзери дають інциденту назву, таку як «Soccer: Liverpool vs Manchester City.» або «presidency of Trump», а також тимчасовий додатковий проміжок. Коли юзери припиняють введення запиту, розроблений сервіс зберужує інциденти та розпочинає знаходження твітер повідомлення, що належають запиту.

У даній імплементації знаходяться твітер повідомлення тільки за цільовим словом, коли юзер вперше заносить цільове слово у додаток, що не є обмежуванням жорстким. Для одержання історичного вигляду інциденту з його цільових слів, юзер може постійно колекціонувати зразки отриманих твітер повідомлень і індексувати послідовно кожне цільове слово, коли юзери починають відслідковувати їх.

Послідовна діаграма утворення запиту представлена та розроблена у додатку.

4.2 Сентиментальне аналізування

Алгоритм аналізування сентиментів твітер повідомлень призводить до одержання негативних та позитивних класів. Алгоритм застосовує Bayes класифікувач, навчений з використанням unigram функцій. Доречно розробити навколо ідеї алгоритм, Go описано-генеруючогих тренінг наборів для негативних та позитивних класів з використанням твітер повідомлень з сумними та щасливими смайликами.

Найчастіше, алгоритми аналізування емоцій у форматі tweet-by-tweet оцінюються, застосовуючи показники стандартного відзивання і точності. Проте насправді такі алгоритми найчастіше застосовуються у отриманій формі. Звичні

інструменти відображення отриманих емоцій можуть генерувати погляд упереджений. Але сприяння системи покращує дане зміщення.

Класифікувач прогнозовує випадковість того, що твітер повідомлення – це частина класу негативного і випадковість того, що він є частиною класу позитивного. Сильну думку виражають не всі твітер повідомлення, тому необхідний ступінь довіри до класифікувачів, прогнозування нейтральної категорії яких нижче. Підвищення ступеню для класу на гарно навчений класифікувач на збільшення строгості та зменшення відзивання класифікувача. Наприклад, можна збільшити строгість на позитивному класі шляхом підвищення ступеня, для якого приймемо твітер повідомлення з негативними встановленими значеннями.

Негативні та позитивні класифікувачі матимуть різноманітні критерії строгості, відзивання, коли налаштовуються їхні ступені. Менш за все крива строгості, відзивання буде одинаковою для всіх класів. Це більш за все призведе до того, що всі класи матимуть різноманітні значення відзивання з тією ж строгістю, що можна зустріти на рисунку 4.1.

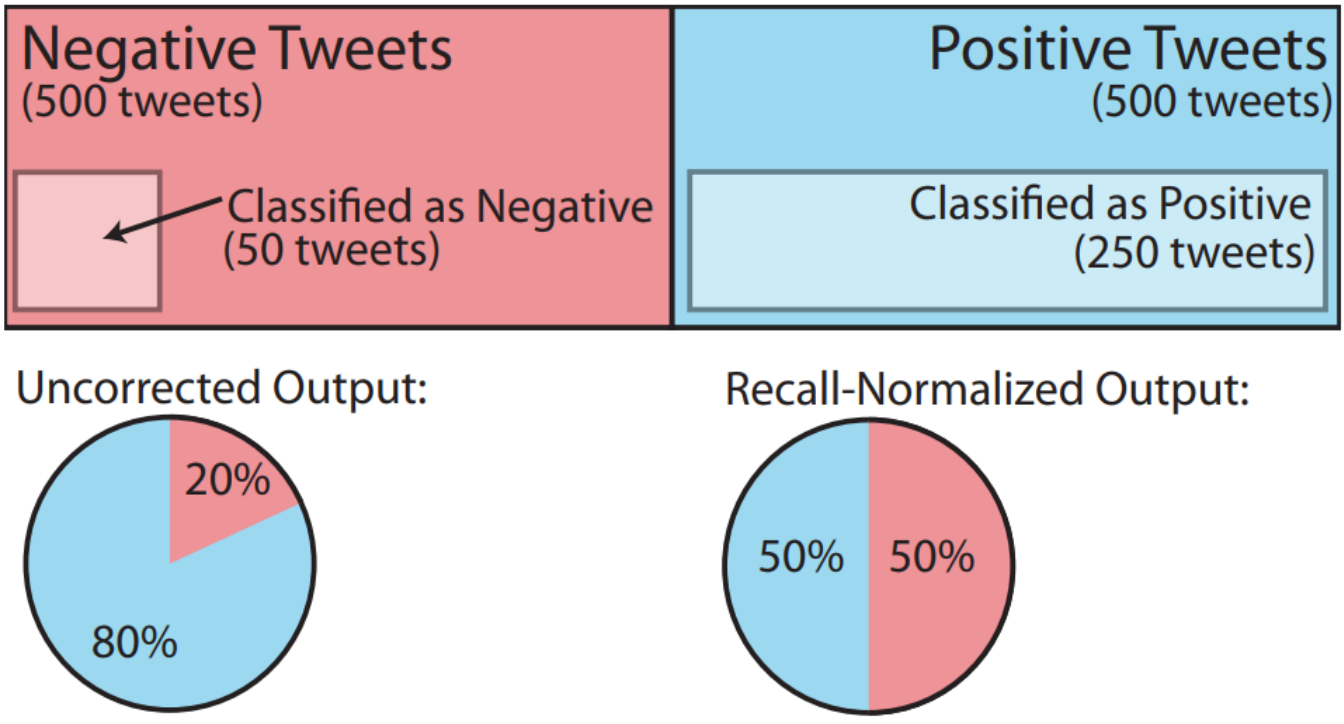

Рисунок 4.1 – Приклад результатів аналізування повідомлень з мережі Twitter

Класифікувачі негативного та позитивного відчуття часто мають різну потужність. Припустимо, що як негативний, так і позитивний клас уміщують 1320 повідомлень твітер. Якщо негативний класифікувач є ліберальним і тільки відшуковує 35 твітер повідомлень , тоді як позитивний класифікувач більш ліберальний та ідентифікує 331, пропорції відносні залишаються передбачуваними. Хоча, якщо застосувати оцінювання відзивання класифікувача, як цифру в діапазоні [1 ... 2] , то точніше візуалізування утворюється без проблем.

Це доводить, що будь-який класифікувач може ліберально ігнорувати твітер повідомлення, а інший може сприйняти їх. Беручи до уваги приклад, де фігурує 330 негативних та позитивних повідомлень твітер, але позитивний класифікувач має відзивання 2, тоді як негативний класифікувач має відзивання 5, то негативний класифікувач відшуковує у 6 разів більше негативних твітер повідомлень як позитивні у сукупності. В разі не уникнення цієї проблеми, юзер побачить отримане візуалізування, яке грубо переоцінює кількість негативних твітер повідомлень .

Розв’язання такої проблеми полягає у тому, щоб нормалізувати отримані числа. Спочатку налаштовуємо обидва класифікувачі так, щоб вони мали еквівалентні точні значення на тестовому наборі даних. Потім вимірюємо їх відзивання на цей набір. Коли підраховуємо кількість позитивних твітер повідомлень , то розбиваємо рахунок на позитивне відзивання, а також нормалізуємо негативний підрахунок.

Припустимо, якщо відомо, що згадування є 2, і спостерігалось 35 позитивних твітер повідомлень , то оцінюємо 35/0.2 = 700 негативних твітер повідомлень . Ця корекція передбачає, що характеристики точності, відзивання алгоритму аналізування сентиментів на практиці подібні до аналізування на наборі даних тесту. Оскільки агреговані емоції важливі у багатьох областях, вважається, що дослідження принципового розв’язання отриманої проблеми є необхідним у сентиментального аналізування сфері.

Розроблена система забезпечує інтерфейс та інфраструктуру над потоком твітер повідомлень. Коли утворюється подія, системою шукаються цільові словосполучення для такої інциденту, застосовуючи API Streaming Twitter. Вона зв’язує повідомлення твітер, що вміщують належні цільові словосполучення з

подіями, вираховує обсяг повідомлень з плином часу і видаляє метадані, такі як адрес URL та смайли. Зберігає метадані гео локації шляхом збирання тегів геолокації твіту, або, якщо це неможливо, спробує перетворити поле місцезнаходження юзера у пару довгота та широти з використанням служби геолокації.

Отримані повідомлення твітер зберігаються до Mongo DB бази даних з використовуючи серверну інфраструктуру написаної з використанням Spring Boot Framework. Твітер повідомлення індексуються за цільовими словами, які відповідають запиту потоку. Коли юзер запитує нове аналізування, то розроблена система відшуковує належні твітер повідомлення, застосовуючи індекси на цільове слово та часові позначки.

Для візуалізації анотованих діаграм та шкал часу застосовується API Visualization Google і також утворюється мапа з використанням API Maps Google.

4.3 Хронологія твітер повідомлень

Відразу як юзери утворили запит, вони зможуть відслідковувати запит у режимі real time, перейшовши на веб-сторінку, яку система утворює для інциденту. Головний інтерфейс системи – це інформаційна панель, яка узагальнює інциденту з часом. На інформаційній панелі відображається шкала часу для такої інциденту, текстовий сигнал, обраний з інциденту, графік оглядових емоцій та погляд на карту, що відображає емоції місцезнаходження та чіткості.

Хронологія інциденту повідомлює про збільшення активності твітер повідомлень за об’ємом. Чим більше твітер повідомлень , які належать запиту на протязі встановленого часового періоду, тим більше значення на осі Y на шкалі часу для такого ж періоду. Отже, коли велика кількість людей створюють тівіти на певну тематику, наприклад, що президентом став Дональд Трамп, то шкала часу збільшуватиметься.

Ідентифікаційний алгоритм вершин системи, відображений далі у дисертації. Він автоматично вираховує ці вершини та зображує їх як вершини у користувацькому інтерфейсі. Вершини утворюються у вигляді прапорців у шкалі часу і утворюються

ліворуч від шкали часу у поєднанні із утвореними автоматично цільовими термінами, які часто утворюються у твітах під час створення вершини. Наприклад, на малюнку, система автоматично доводить одну з цілей у футбольній грі як пік «F» та анотує її ліворуч із репрезентативними термінами у твітах, як 0:4 – новий рахунок.

Юзери можуть виконувати текстовий пошук у цьому списку цільових термінів, щоб знайти певний пік. Щоб візуально масштабуватися від коротких подій до довго триваючих запитів, система агрегуватиме твітер повідомлення на часовій шкалі з деталізацією за хвилиною, годиною або днем залежно від часу, який переглядає юзер.

Часова лінія також надає метод фільтрування твітер повідомлення у веб інтерфейсі, коли юзер натискає на пік або інші елементи інтерфейсу твітер повідомлень, список карти чи посилання.

Тепер визначаємо, чи підхід до автоматичного визначення подій відповідає людським інтуїціям про правильну поведінку. Щоб провести оцінку, важливо зібрати твітер повідомлення з трьох футбольних ігор і одного місяця землетрусів. Для одержання істинну картину про футбольні дані, один дослідник коментувати основні інциденту у футбольній грі, застосовуючи ігрові відео та веб-підсумки гри, не дивлячись на твітер повідомлення. Для істини текстових даних про землетрус, важливо зібрати дані з Геологічної служби США про великі землетруси на протязі періоду часу.

Доцільно перевірити, скільки істинних подій виявив алгоритм – точність, і скільки подій розпізнано правильно, що є повнотою. Для визначення подій використовувалось порогове обмеження за замовчуванням. Можемо компенсувати точність та повноту, коригуючи цей поріг, але доведено, що треба використати єдине значення. Результати наведено у таблиці 4.1.

Алгоритм має високу повноту, знаходячи всі основні твітер повідомлення про землетруси та більшість футбольних подій. На рисунку 3 показані кілька часових проміжків на основі подій, які ідентифікував алгоритм.

В ході дослідження проаналізовано три футбольні ігри. Алгоритм ідентифікації піку відшуковує тільки 1 помилково позитивний. Алгоритм не міг знайти запит «Жовті картки", але не виявлено будь-якого спадання використання Twitter. Цей

алгоритм іноді позначав попередню обговорення, як повторне. Результати аналізування зображено на рисунку 4.2.

Таблиця 4.1 – Приклад результатів аналізування спортивних подій

Data Source	Precision	Recall
Soccer Events	$\frac{17}{22} \Rightarrow 77\%$	$\frac{17}{22} \Rightarrow 77\%$
Soccer Events & Discussion	$\frac{21}{22} \Rightarrow 95\%$	$\frac{21}{26} \Rightarrow 81\%$
Major Earthquakes	$\frac{6}{44} \Rightarrow 14\%$	$\frac{5}{5} \Rightarrow 100\%$
All Earthquakes	$\frac{29}{44} \Rightarrow 66\%$	N/A
All Earthquakes & Discussion	$\frac{39}{44} \Rightarrow 89\%$	N/A

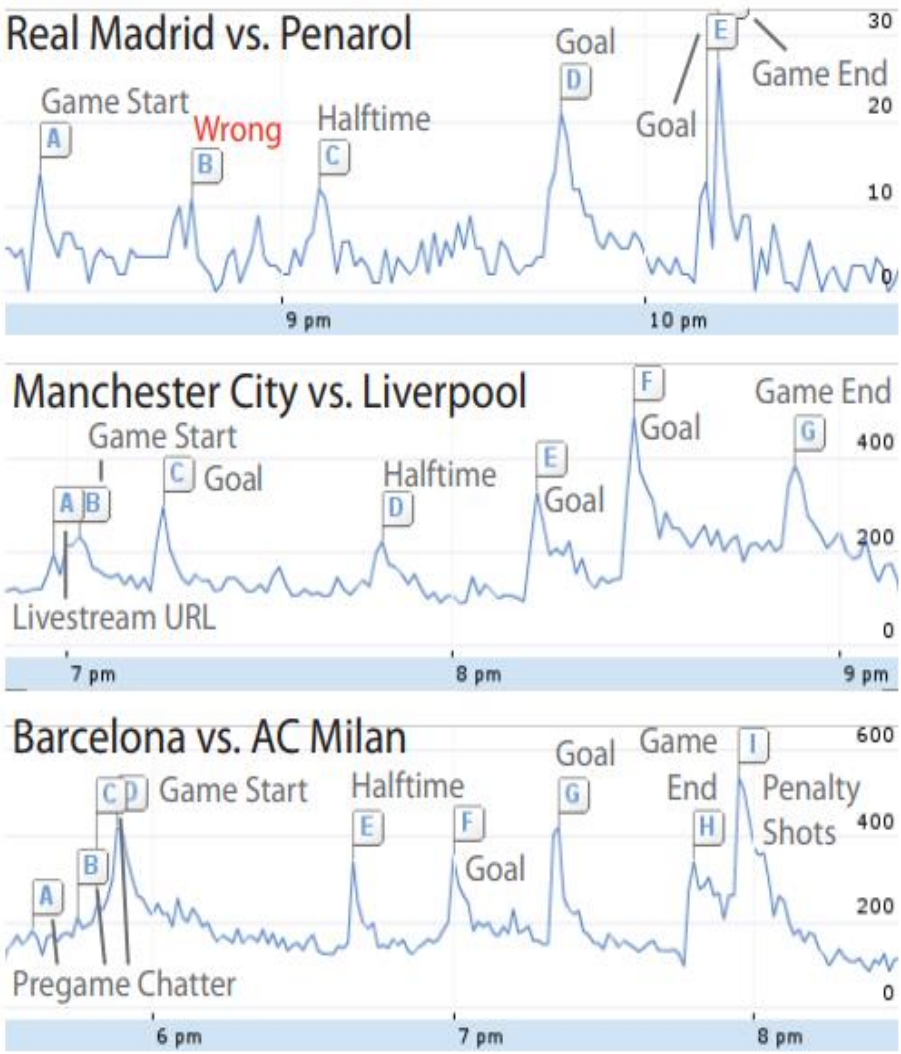

Рисунок 4.2 – Результати аналізування спортивних подій

Точність залежить від характеру активності, якщо вона являлась високою для спорту, то для природніх катаклізмів виявлено кілька хибних спрацьовувань, оскільки алгоритм також позначав землетруси із незначними показниками. Якщо юзер включить оцінку всіх подій то, точність підвищиться з 15% до 67%. Алгоритм як і раніше виробляє неправильні негативи, оскільки мережа Twitter зменшує кількість юзерів, які говорять про природній катакліх та діляться важливою інформацією про відшкодування постраждалого майна.

Якщо враховувати всі обговорення, пов'язані зі природнім катаклізмом, то строгість підвищиться до 92%. Точно так само, включаючи обговорення, пов'язані з футболом, наприклад, бесіди, що призведе до 97% строгості у футбольному наборі текстових даних.

Залишок неправильних аналізів походить з великих об’ємів тематик, які зустрічались, щоб згадати слова повязані з природнім катаклізмом або футбольним терміном.

4.4 Ідентифікація подій

Тепер звернемося до алгоритмів, які застосовуються для ідентифікації та відображення кінцевого результату аналізування. Цільовим технічним внеском у цій роботі є визначення часових піків у частотному діапазоні. Під час реалізації поставлених технічних завдань розроблено наступний алгоритм. В алгоритмі описано версію пікового ідентифікування. Псевдокод алгоритму пошуку піків зображено на рисунку 4.3.

```
function find_peak_windows(C):
windows = []
mean = C_1
meandev = variance(C_1, ..., C_p)

for i = 2; i < len(C); i++ do
    if |C_i − mean| / meandev > τ and C_i > C_{i−1} then
        start = i − 1
        while i <len(C) and C_i > C_{i−1} do
            (mean, meandev) = update(mean, meandev, C_i)
            i++
        end while
        while i < len(C) and C_i > C_start do
            if |C_i − mean| / meandev > τ and C_i > C_{i−1} then
                end = − − i
                break
            else
                (mean, meandev) = update(mean, meandev, C_i)
                end = i++
            end if
        end while
        windows.append(start, end)
    else
        (mean, meandev) = update(mean, meandev, C_i)
    end if
end for
return windows

function update(oldmean, oldmeandev, updatevalue):
diff = |oldmean − updatevalue|
newmeandev = α*diff + (1−α)*oldmeandev
newmean = α*updatevalue + (1−α)*oldmean
return (newmean, newmeandev)
```

Рисунок 4.3 – Алгоритм пошуку пікової активності

4.5 Визначення належних твітер повідомлень

Оскільки зазвичай фігурує ширша кількість твітер повідомлень , ніж та яка може бути утворена юзером для будь-якого об'єкта, додаток старається визначити належні твітер повідомлення для юзера у списку належних твітер повідомлень. З огляду 7-термінованої етикетки для кожної ідентифікованого інциденту, оцінюються твітер повідомлення за обсяго івентів з subevent міткою, що містить твітер повідомлення. Так як ретвіт повідомлення вміщують повторюваніслова, що і твітер

повідомлення, то вдвічі збільшується час розрахунку твітер повідомлень, які повторюються.

4.6 Веб інтрефейс користувача

Юзер керує системою використовуючи інтерфейс веб сервісу Facebook Messenger. На рисунку 4.4 зображено типовий випадок взаємодії з системою.

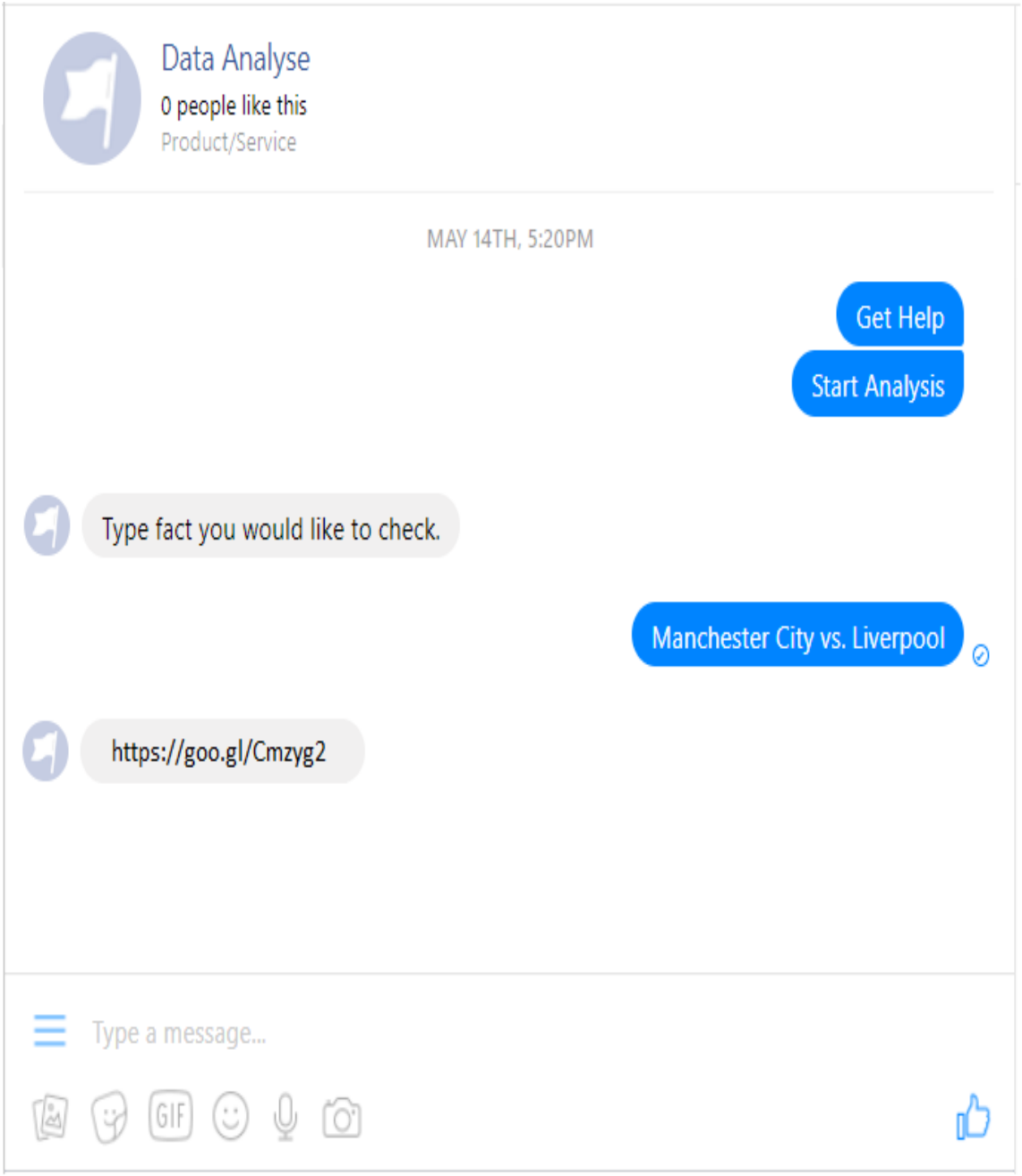

Рисунок 4.4 – Інтерфейс сервісу Messenger Facebook

Попередньо запрограмований бот інтерфейс надсилає юзеру запити, щоб він зміг відправити факт для аналізування. Юзер відправляє текстове повідомлення з вмістом «Liverpool vs. Manchester City», та отримує гіпер посилання на інформаційну панель, де зображено результат запиту.

Інформаційна панель – це головний інтерфейс системи, головними візуальними компонентами якої є карта та шкала часу.

З часом система узагальнює інциденти. На панелі інформації показується часова шкала для такого інциденту чи пошукового запиту, текстовий сигнал, який було обрано з інциденту, графік отриманих емоцій, що видно на рисунку 4.5. А також на карті відображаються гео-локації, звідки відправлено твітер повідомлення, що видно на рисунку 4.6.

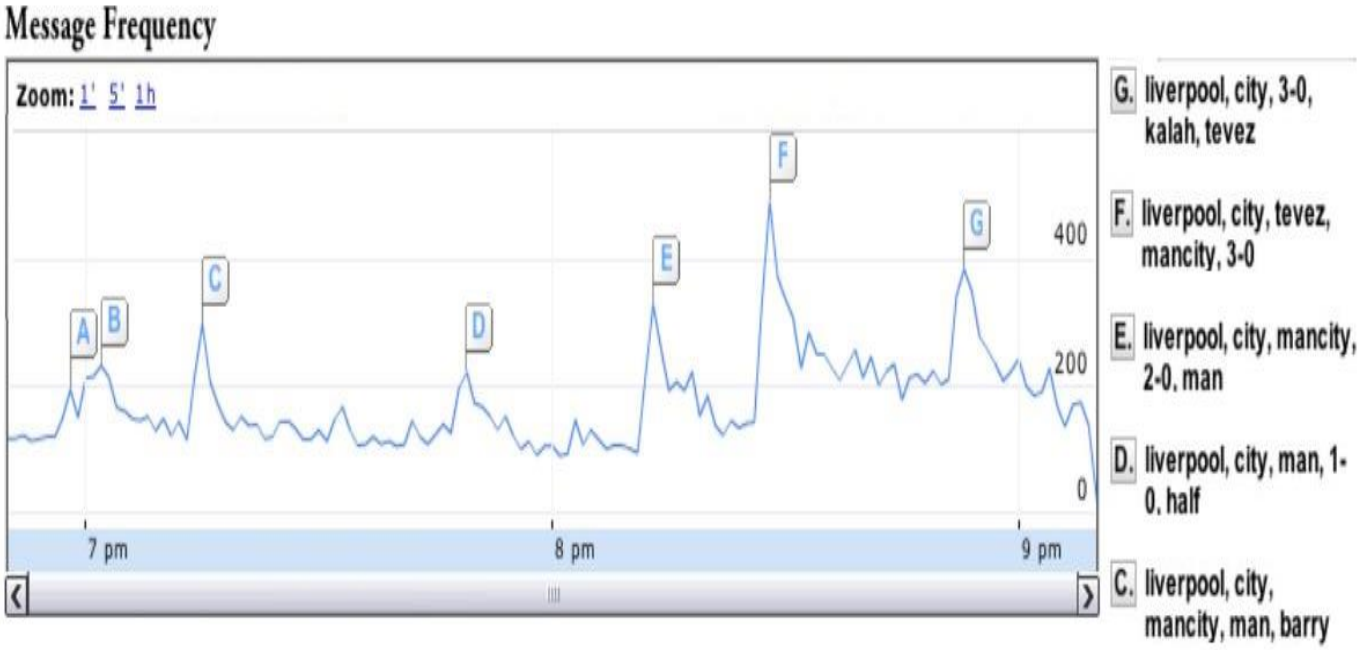

Рисунок 4.5 – Інформаційна панель з часовою шкалою

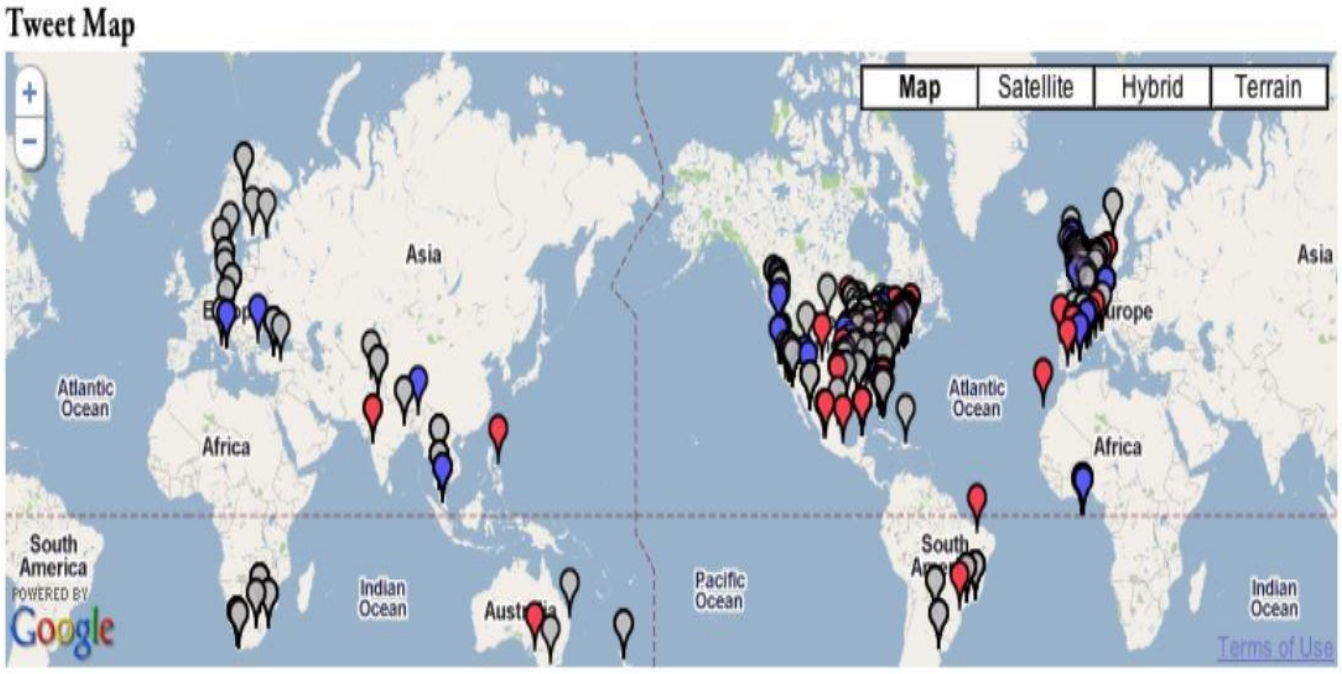

Рисунок 4.6 – Карта з відображенням географічних точок юзерів

4.7 Тестування системи

Проведено тестування системи на протязі всього циклу розробки. Вдалось проаналізувати та імплементувати декілька типів тестування на різноманітних стадіях розробки. Також впроваджено наступні типи тестування:

- юніт тестування;
- інтеграційне тестування;
- функціональне тестування;
- мануальне тестування.

Юніт тестування доцільно використовувати та імплементувати в зв’язку з тим, щоб оцінити роботи вихідного коду програми, а саме окремих модулів. Таке тестування впроваджується на найвищому ступені, відокремлюючи модуль, який тестується, від модулів, які залежні і вважаються протестованими, за для ідентифікації помилок тільки у тестованому модулі. Варто покривати тестами сервіси прикладного відкритого інтерфейсу API, сервіси роботи з базою даних, Kafka Stream сервісів призначених для опрацювання текстової інформації із та інші сервіси системи. Покриття тестами юніт є 80% (рисунок 4.7).

Рисунок 4.7 – Графічне представлення покриття системи тестами

Тести інтеграційні імплементовані, для перевірки справності та якості модулів у розгорнутій системі, без використання стаб та мок підходів. На даному етапі оточення тестовий енваромент розгортає робочу систему, у де тестові сценарії запускаються. Тестами покриття становить 15% вихідного коду (рисунок 4.7).

Тести функціональні розроблені для end-to-end тестування повторюючи сценарії використання системи. Таке тестування проводиться для перевірки роботи програмного продукту. Тестове покриття становить 5% вихідного коду (рисунок 4.7).

Тести юніт, інтеграційні та функціональні запускаються кожного разу якщо код у репозиторії змінюється. Дана можливість імплементована з використанням технології безперервної інтеграції і доставки системи. Швидкість розроблення та якість системи зросла за рахунок такого підходу та зменшувався ризик поломки готових до використання модулів.

Мануальне або ж ручне тестування проводились в робочій системі у розгорнутому тестовому оточенні. Найважливіші тестові випадки доцільно пов’язати з отриманням результатів аналізування та порівнянням їх з еталонними значеннями.

4.8 Висновки до розділу 4

Отже, у результаті розроблення програмного комплексу, вдалось отримати робочий прототип системи збирання та аналізування текстових даних з соціальних мереж. Робота розгорнутої системи демонструє повний цикл оброблення інформації розпочинаючи зі збирання текстових даних з соціальної мережі Twitter, їх попереднього оброблення, фільтрації та семантичного аналізування з використанням методів оброблення натуральних мов, а також агрегації та збереження текстових даних до сховища для подальшого використання у аналізуванні.

Налаштований процес безперервного тестування та розгортання мінімізовує час розгортання системи, а також забезпечує якість продукту продовж його життєвого циклу. Графічний інтерфейс системи забезпечує зручне використання цільовими юзерами – аналітиками, що мінімізовує витрати часу на навчання та забезпечує низький поріг для початку роботи з нею.

5 РОЗРОБКА СТАРТАП ПРОЕКТУ

Стартап – це бізнес-структура, що працює на основі руйнівних інновацій, утворених для розв’язання проблеми шляхом доставки нового продукту чи послуги у умовах крайньої невизначеності [19].

Багато підприємців та відомих ділових магнатів визначають стартап як культуру та менталітет побудови бізнесу на основі інноваційної ідеї розв’язання критичних больових моментів.

5.1 Опис ідеї

Основна ідея полягає у забезпеченні швидкого та якісного збирання текстових даних з вибраних соціальних мереж. Система збирання та аналізування текстових даних отриманих з соціальних мереж реалізовує методи швидкого збирання та безперервної асинхронного оброблення текстових даних, застосовуючи інструменти оброблення натуральних мов, надає легку та швидку інтеграцію з належними сервісами, які потребують попередньо підготовані дані.

Методи збирання та аналізування текстових даних:

– Інтеграція з соціальними мережами: сюди входить підтримка різноманітних протоколів інтеграції належно до вибраної соціальної мережі. Базова реалізація підтримує інтеграцію з найпопулярнішими мережами такими як Twitter з використанням Twitter Stream API та Facebook з використанням Facebook Graph API.

– Потокове оброблення текстових даних: використання методів потокової оброблення даних для безперервного та швидкого одержання результатів роботи системи з мінімальними часовими витратами.

– Попереднє оброблення текстових даних: перетворення текстових даних у стандартизований формат, для покращення результатів подальшого аналізування тексту.

– Фільтрація шумів: використання розглянутих методів визначення шумності тексту, для покращення якості вихідних даних.

– Сентиментальне аналізування: розбиття текстових даних на сентиментальні групи.

–Збереження текстових даних: налаштування інфраструктури для швидкої інтеграції системи з обраними сховищами текстових даних. Базова реалізація підтримує такі сховища як MongoDB – NoSQL, а також PostgreSQL – RDBS.

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки використання	Вигоди для юзера
Зміст ідеї є розроблення системи швидкого безперервного збирання та аналізування текстових даних з соціальних мереж для її подальшої інтеграції з системами детального аналізування.	Може застосовуватися як приватними так і публічними проектами, для яких основною вимогою є швидка інтеграція та якісний збір даних.	За рахунок швидкої інтеграції з даною системою клієнти можуть у короткі строки розпочати збір даних з соціальних мереж для одержання готових даних для їх подальшого аналізування. Таким чином, клієнти у короткі строки зможуть розпочати свою діяльність, що збільшує їхні прибутки.

Після пошуку та аналізування можливих конкурентів, що займаються збором та аналізуванням текстових даних, вдалося знайти декілька конкурентів, серед яких виділяються Matillion та Spring Boot Cloud Data Flow. Використання наявних на ринку систем та рішень не забезпечить цільовому проекту швидкого та якісного збирання текстових даних з соціальних мереж, так як наявні рішення є уніфікованими і вимагають безпосереднього налаштування інтеграції. Проте інтеграція з соціальними

мережами, не є найскладнішим завданням у даному питанні. Основна складність залишається за специфічним аналізуванням текстових даних утворених реальними юзерами.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент 1	Конкурент2			
1.	Базова інтеграція з популярними соціальними мережами	Інтеграція з Twitter та Facebook	відсутня	відсутня	-	-	присутня
2.	Визначення шумності тексу	присутнє	відсутнє	відсутнє	-	-	присутня
3.	Вартість впровадження	низька	висока	середня	складність імплементації	-	-
4.	Технології розробки	Java, Spring Boot Framework, Spring	Користувацький інтерфейс у поєднанні	Java, Spring Framework	-	-	Новітні технології

		Boot, Tomcat	з Python скріптами				

На відміну від існуючих рішень, розроблювана система включає у себе послідовність методів оброблення, які забезпечують високу якість отриманих даних.

Розробка є інноваційною, тому що включає у себе використання високоефективних методів інтеграції з соціальними мережами, потокову асинхронну обробку даних, попередню обробку текстових даних, фільтрацію текстового шуму, сентиментальне аналізування, якщо потрібно, а також зручний для подальшої інтеграції з системою підхід до агрегування та збереження вихідних даних.

До прямих конкурентів можна віднести такі системи:

– конкурент 1 – система Matillion;
– конкурент 2 – система Spring Boot Cloud Data Flow.

5.2 Технологічний аудит ідеї

Перед тим, як замовити програмний продукт, компанії хочуть знати, що обраний ними виробник має необхідну інфраструктуру, системи, процеси та можливості для його виконання. Вони також повинні забезпечити постійне додержання місцевих та міжнародних стандартів якості та безпеки у всіх їх міжнародних постачальників, а також у процесі виробництва та доставки.

Тому для вибору найбільш слушних технологій доцільно провести аналізування ринку технологій, які задовольняють вимоги до системи.

Обрані базові технології допоможуть з легкістю реалізувати систему. Мова програмування Java забезпечить крос-платформенність отриманої системи, завдяки особливості віртуальної машини Java, яка виконує вихідний код. Тому подальший вибір хостинг системи не являється обмеженим. Також дана мова програмування є однією з найпопулярніших, що забезпечить легкий набір висококваліфікованих програмних інженерів.

Основний фреймворк для розроблення – Spring Boot Framework. Дана технологія є найчастіше вживаною та популярної серед Java розробників. Вона має безліч переваг над іншими. Даний фреймворк є безкоштовною open source технологією, що забезпечує безперешкодне використання, а також постійну підтримку спільнотою Spring Boot [20]. Результат технологічного аудиту представлено у таблиці 5.3.

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології імплементації	Наявність технологій	Доступність технологій
1	Основний фреймворк для розробки	Spring Boot Framework	+	+
2	Основний фреймворк для розробки	Asp.net Core	+	+
3	Мова програмування	Java	+	+
4	Мова програмування	Scala	+	+
5	Мова програмування	Groovy	+	+
6	Мова програмування	C#	+	+
7	Хостинг	Digital Ocean	+	+
8	Хостинг	AWS EC2	+	+
9	Система CI/CD	Jenkins CI	+	+
10	Система CI/CD	Travis CI	+	+
11	Репозиторій	GitHub	+	+

12	Репозиторій	GitLab	+	+
13	Система збірки	Maven	+	+
14	Система збірки	Gradle	+	+
Обрані технології імплементації ідеї проекту: Spring Boot Framework, Java, Digital Ocean, Travis CI, GitHub, Maven				

Наступним кроком являється визначення ринкових можливостей стартап - проекту, вони наведені у таблиці 5.4.

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	2
2	Загальний обсяг продаж, грн/ум.од	10
3	Динаміка ринку (якісна оцінка)	зростає
4	Наявність обмежень для входу (вказати характер обмежень)	немає
5	Специфічні вимоги до стандартизації та сертифікації	відсутні
6	Середня норма рентабельності у галузі (або по ринку), %	76%

Аналізування ринку впровадження, де розгортатиметься проект, показує, що обсяг конкурентів не великий, але наявні учасники вже мають своїх клієнтів, що зменшує кількість потенційних юзерів [21].

Так як, сфера збирання згенерованих юзерами даних, а також їх аналізування є досить актуальною на сьогодні, то попит на програмний продукт подібного роду є

доволі високим. Такі твердження доводять, що розробка та вивід такого програмного забезпечення є рентабельним та вигідним.

Для вдалого та швидкого провадження даного продукту на ринку важливо визначити детальну стратегію розроблення та впровадження. Перш за все, правильний і якісне аналізування конкурентів, забезпечить миттєві успішні показники на початку функціонування. Детальне аналізування існуючих конкурентів представлено у таблиці 5.5.

Можливі різноманітні підходи до запуску проекту та виходу на ринок, а саме:

– постачати послуги, як допоміжний сервіс;

– пропонувати безпосереднє інтегрування з сервісами, які потребують дані послуги, з подальшою підтримкою індивідуального рішення.

Групи цільових клієнтів та базові вимоги до споживачів, які виокремлюватимуть розроблювану систему відображено у таблиці 5.5. Цільовий обсяг проектів є доволі великим, з його допомогою можна надійно управляти проектом та впроваджувати його.

Таблиця 5.5 – Характеристика потенційних клієнтів стартап – проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різноманітних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1		Приватні компанії	Приватні програмні рішення, які потребують швидкої інтеграції	Швидка інтеграція системи з робочими системами клієнтів
2		Open Source проекти	Відкритий програмний код для	Доступність вихідного коду для

			загального використання	самостійної інтеграції
3		Наукові організації	Програмні рішення для проведення наукових досліджень	Доступність вихідного коду для самостійної інтеграції

При розробці програмного забезпечення, його проектування чи вже під час його роботи, можна зіштовхнутись з так званими факторами загроз. Якщо таких факторів зустрічається багато, то існування проекту може є під загрозою.

Фактор загроз та фактор можливостей – це суттєві фактори, що можуть впливати на програмний продукт.

Ймовірні фактори загроз при запуску проекту на ринку перелічено у таблиці 5.6. Важливо подбати про те, щоб такі загрози вдалося ліквідувати раніше, ніж проект почне свою діяльність.

Таблиця 5.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Кваліфікація учасників команди	Низький ступінь знань працівників, а також не значний досвід у області розроблення систем збирання та аналізування даних	Утворення системи відбору кваліфікованих працівників, що підвищить ступінь найманих кадрів
2	Конфіденційність даних системи	Розкриття особливостей роботи системи та	Залучивши належних спеціалістів, таких як юристи, розробити та укладати

		функціонування компанії вцілому	договори, які зменшать ризик витоку конфіденційних даних
3	Недоступність системи	Відключення робочої системи у зв’язку з технічними неполадками	Налаштування систем моніторингу для запобігання раптового відключення системи
4	Поява конкуруючих рішень	Утворення конкуруючих систем	Розвиток системи для залучення більшого обсягу клієнтів

Проблеми, які система старається вирішити чи подолати – це фактори можливостей. До таких факторів можна віднести швидкодію, навантаження, високу точність чи віртуалізування. Наступним етапом є розгляд факторів можливостей, що відображено у таблиці 5.7.

Таблиця 5.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Попит на систему	Можливість підвищення кількості юзерів	Утворення рекламних кампанії. Утворення маркетингового відділу для підвищення попиту
2	Висока продуктивність програмного забезпечення	Можливість покривати потреби продуктивності потенційних та існуючих юзерів	Контроль якості розроблення програмного забезпечення. Залучення незалежних спеціалістів для якісної оцінки роботи системи. Проведення навантажувального тестування

3	Публічний проект	Можливість розміщення рекламного контенту	Утворення умов для розміщення реклами для заохочення нових клієнтів

Приступимо до ступеневого аналізування конкуренції на ринку. Для аналізування доцільно обрати такі цільові характеристики як «Особливості конкурентного середовища», «В чому проявляється дана характеристика» та «Вплив на діяльність підприємства». Результати аналізування описано у таблиці 5.8.

Ступеневе аналізування конкуренції на ринку дозволяє зробити більш детальне аналізування очікуваних конкурентів перед початком діяльності компанії та виводу програмного продукту на ринок[21].

Таблиця 5.8 – Ступеневе аналізування конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Тип конкуренції – ринкова конкуренція	Присутність конкурентів на ринку	Постійний розвиток програмного продукту
2. За рівнем конкурентної боротьби – міжнародна	Клієнти є міжнародними компаніями чи організаціями	Підтримка локалізації та інтернаціоналізації системи. Основною мовою зробити англійську
3. За галузевою ознакою – міжгалузева	Система може використовуватись у різноманітних галузях	Бути готовим швидко реагувати на запити клієнтів з різноманітних галузей

4. Конкуренція за видами товарів – товарно-видова	Вид наданих послуг є товарно-видовими	Реалізація цільових вимог. Одержання максимальних показників продуктивності
5. За характером конкурентних переваг – якісна	Якість наданих послуг є основним пріоритетом та перевагою даної системи	Утворення умов для запровадження методів та технік, які збільшать якість наданих послуг
6. За інтенсивністю – марочна	Прояв марочної інтенсивності	Бути конкурентною компанією з впровадженням передових технологій

Проведемо аналізування за методом п’яти сил конкуренції Майкла Портера [22]. Результати аналізування відображено у таблиці 5.9.

Таблиця 5.9 – Аналізування за методом п’яти сил конкуренції Майкла Портера

Складові аналізування	Прямі конкуренти у галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Відсутні	Присутні	Відсутні	Рішення, які потребують інтеграції з соціальними мережами для збирання текстових даних	Системи ETL такі як Matillion чи Spring Boot Cloud Data Flow

Висновки:	На даний момент безпосередніх конкурентів не знайдено, але є подібні проекти	Наявність існуючих конкурентів на ринку, не заважає проекту для виходу та успішної роботи	Постачальники подібних послуг не диктують умови роботи системи на ринку	Потенційні клієнти визначають напрямок та розвиток роботи системи. Основними вимогами є доступність даної системи	Жодних обмежень не виявлено

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Можливість інтеграції з різними соціальними мережами	Досягається за рахунок попередньо спроектованих та розроблених конекторів, які забезпечують швидку інтеграцію з різними соціальними мережами, у залежності від їх протоколу інтеграції
2	Можливість попередньої оброблення даних	Досягається за рахунок використання технологій оброблення натуральних мов, включаючи ці методи у процес збирання даних
3	Можливість фільтрування шумних даних	Досягається за рахунок використання розроблених алгоритмів фільтрації шумного тексту згенерованого юзерами

4	Можливість агрегування та збереження даних у різноманітних типах баз даних	Досягається за рахунок попередньо закладеного у архітектуру сиситеми рішення розширення чи зміни типу сховища даних

Результати аналізування конкуренції ринку за М. Портера дозволяють зробити висновок, що розгортання проекту цілком можливий та він вважається успішним у разі виконання встановленого плану [23].

Обґрунтування факторів конкурентоспроможності (таблиця 5.10) встановлює чіткі умови та цілі для одержання конкурентоспроможного програмного продукту. Для одержання максимального прибутку важливо дотримуватись визначених факторів.

Для визначення сильних та слабких сторін проекту доцільно провести порівняння проекту з проектами потенційних конкурентів та їх аналізування. Для цього важливо ввести рейтингове порівняння програмних продуктів [24]. Результати порівняння зображено у таблиці 5.11.

Таблиця 5.11 – Порівняння сильних та слабких сторін проекту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів						
			–3	–2	–1	0	+1	+2	+3
1	Можливість інтеграції з різними соціальними мережами	18			К2	К1			+
2	Можливість попередньої оброблення даних	19		К2	К1				+
3	Можливість фільтрування шумних даних	20		К1	К2				+
4	Можливість агрегування та збереження даних у різноманітних типах баз даних	20	К1	К2					+

Таблиця 5.12 – SWOT – аналізування стартап-проекту

Сильні сторони: Використання методів оброблення натуральних мов для попередньої обробки даних. Фільтрування шумних текстових даних, які вдалось згенерувати реальними юзерами	Слабкі сторони: Складна розробка системи. Важке розгортання системи
Можливості: Підвищення кількості юзерів та клієнтів у зв’язку проведення успішних рекламних кампаній	Загрози: Поява конкурентних проектів, які зменшуватимуть кількість клієнтів

SWOT аналізування визначає сильні, слабкі сторони, можливості та загрози, і тому SWOT аналізування – це методика оцінки цих чотирьох аспектів бізнесу [25].

SWOT аналізування важливо застосовувати, щоб максимально використати все, що у є на даний момент, з найкращими перевагами організації чи стартапу. Це допомагає знизити шанси на невдачу, зрозумівши, чого не вистачає, і усунути небезпеки, які у іншому випадку могли б зненацька з’явитись. Тому проведено SWOT аналізування, результати якого представлені у таблиці 5.12.

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Випадковість одержання ресурсів	Строки імплементації
1	Реалізація системи на базі Spring Boot Framework та Java, інтеграція з Twitter та Facebook, попереднє оброблення даних з використанням методів	Висока	26-32 місяців

	оброблення натуральних мов, фільтрація шумних текстових даних		
2	Реалізація системи без інтеграції з соціальними мережами	Низька	16-20 місяців
3	Реалізація системи без попередньої оброблення даних	Середня	18-22 місяців
4	Реалізація системи без фільтрації шумних текстових даних	Середня	18-24 місяців
5	Реалізація системи з інтеграцією з Twitter та впровадження попереднього оброблення даних	Середня	20-26 місяців

Технологічний аудит системи закінчуються порівнянням альтернатив ринкового впровадження стартап проекту, що описано у таблиці 7.13. Тут розраховано приблизний час на реалізацію альтернативних варіантів системи [26].

5.3 Розроблення ринкової стратегії проекту

Таблиця 5.14 – Цільова групи споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит у межах цільової групи (сегменту)	Інтенсивність конкуренції у сегменті	Простота входу у сегмент
1	Приватні компанії	Споживачі мають потребу та цілком готові	Високий попит	Можлива висока	Низька

		сприйняти продукт		конкуренція з плином часу	
2	Open Source проекти	Споживачі мають потребу та цілком готові сприйняти продукт	Середній попит	Досить низька конкуренція	Середня
3	Наукові організації	Споживачі мають потребу та цілком готові сприйняти продукт	Низький попит	Відсутня конкуренція	Середня
Які цільові групи обрано: приватні компанії, open source проекти та наукові організації					

Для одержання успішного продукту важливо виділити цільову групу юзерів системи. Для цього проведено аналізування потенційних юзерів у таблиці 5.14.

Таблиця 5.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Цільові конкурентоспроможні позиції належно до обраної альтернативи	Базова стратегія розвитку*
1	Реалізація системи на базі Spring Boot Framework та Java, інтеграція з Twitter та Facebook, попереднє	Концентрації	Рішення, які потребують інтеграції з соціальними	Диференціація

	оброблення даних з використанням методів оброблення натуральних мов, фільтрація шумних текстових даних		мережами для збирання та аналізування текстових даних	

Наступною ланкою аналізування є визначення стратегії конкурентної поведінки, що описано у таблиці 5.15. Доцільно визначити альтернативний розвиток проекту його стратегія охоплення, базову стратегію розвитку, а також цільові конкурентоспроможні позиції. Дане аналізування основане на результатах отриманих з таблиці 5.13.

Після визначення базової стратегії розвитку важливо визначити стратегію конкурентної поведінки (таблиця 5.16).

Таблиця 5.16 – Визначення стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи шукатиме компанія нових споживачів, або забирати існуючих у конкурентів?	Чи копіюватиме компанія основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Ні	Так	Жодних	Фронтальний наступ

Визначення стратегії позиціонування дозволяє встановити вимоги до товару та цільової аудиторії. Також визначити базову стратегію розвитку, а також цільові конкурентоспроможні позиції стартап-проекту.

Таблиця 5.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Цільові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
1	Немає	Диференціація	Швидке та якісне оброблення текстових даних	Швидка інтеграція з вибраними соціальними мережами, якісна підготовка даних до подальшого оброблення, уникнення шумних текстових даних

5.4 Маркетингова програма у стартап-проекті

В даному розділі запропоновано та описано маркетинговий маркетингову стратегію для стартап-проекту.

Для визначення потреб юзерів, вигоди та благ, які пропонує продукт та цільових переваг перед конкурентами, доцільно сформувати таблицю 5.18 з детальним описом вказаних позицій.

До основних потреб юзери віднесено такі особливості системи:

– Інтеграція з соціальними мережами;
– Попереднє оброблення текстових даних з використання NLP;
– Фільтрування шумних текстових даних.

Таблиця 5.18 – Визначення цільових переваг концепцій потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Цільові переваги перед конкурентами (існуючі або такі, що потрібно утворити
1	Інтеграція з соціальними мережами	Готова інфраструктура для швидкої інтеграції з соціальними мережами	Досі не реалізовано у відкритому доступі. Є можливість запровадити даний метод
2	Попереднє оброблення даних з використання NLP	Можливість системи покращити якість даних для їх подальшого оброблення	Досі не реалізовано у відкритому доступі. Є можливість запровадити даний метод
3	Фільтрування шумних текстових даних	Вміння системи знаходження та видаляти чи коригувати шумні текстові дані	Досі не реалізовано у відкритому доступі. Є можливість запровадити даний метод

Для опису трьох рівнів моделі товару сформовано таблицю 5.19. Тут описані послуги та ідея проекту, метод надання послуг, особливості його захисту.

Таблиця 5.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
І. Програмне забезпечення за задумом	Збір та аналізування текстових даних отриманих з належних соціальних мереж		
	Властивості/ характеристики	М/Нм	Вр/Тх /Тл/Е/Ор

	Висока швидкість збирання даних	М	Тх/Тл
	Ефективне попереднє оброблення	М	Тх/Тл
	Фільтрація шумного тексту	М	Тх/Тл
	Програмний продукт пройшов тестування, ISO/IEC 9126 – 1 2013		
	Постачається у форматі розгорнутого сервісу на вибраному хостингу		
	Марка: Bo & Po, AnaliSys, Bo & Co		
	До продажу: програмний комплекс		
	Після продажу: програмний комплекс		
За рахунок чого потенційне програмне забезпечення захищено від копіювання: патент методу збирання та аналізування текстових даних з соціальних мереж, законодавство про авторське право			

Таблиця 5.20. – Визначення меж встановлення ціни

№ п/п	товари-замінники та їх ступінь цін	Товари-аналоги та їх ступінь цін	Цільової групи споживачів та їх ступінь доходів	Верхня та нижня межі встановлення ціни на товар/послугу
1	150 тисяч гривень	Відсутні	Високий ступінь доходів	Від 50 тисяч. гривень за ліцензію

Для визначення з межами встановлених цін, сформовано таблиця 7.20, де детально описаний ступінь цін на очікувану продукції, ціни товарів аналогів, ступінь доходів цільових груп споживачів та межі цін на встановлені послуги.

Після опису цін, можна сформувати та встановити можливі варіанти збуту стартап-проекту. Для цього треба визначити специфіку закупівельної поведінки цільових клієнтів функції збуту, глибину каналу збуду та оптимальну систему збуту. Результати описано у таблиці 5.21.

Останнім пунктом аналізування є визначення концепції маркетингових комунікацій стартап-проекту, що наведено у таблиці 5.22.

Таблиця 5.21. – Формування системи збуту

№ п/п	Цільові клієнти та специфіка їх закупівельної поведінки	Постачальник товару та його функції збуту,	Каналу збуту та його глибина	Система збуту та її оптимальність
1	Замовлення інтеграції або купівля ліцензії у зв’язку з вимогами проекту клієнта	Належність встановлених вимог до системи	Від виробника до споживача	Через маркетингову платформу компанії виробника

Наступним кроком є визначення специфіки поведінки цільових клієнтів та каналів комунікацій, якими користуються цільові клієнти. Також необхідно визначити цільові позиції, обрані для позиціонування, а також завдання рекламного повідомленнятаконцепцію рекламного звернення

Таблиця 5.22 – Концепція маркетингових комунікацій

№ п/п	Специфічна поведінка	Цільові клієнти та їх канали	Обрані для позиціонуван	Рекламне повідомлення	Рекламне звернення та

	цільових клієнтів	комунікацій, якими користуютьс я	ня цільові позиції	та його завдання	його концепція
1	Аналізуванн я пропозиції рішень на ринку та вибір найбільш підходящого	Соціальні мережі, технологічні блоги	Постачання програмного продукту для інтеграції з соціальними мережами	Збільшити зацікавленість клієнтів у даній системі та підвищити попит	Моделювання потенційного ефекту для заохочення майбутніх клієнтів

5.5 Висновки до розділу 5

В розділі описано та проаналізовано розробку стартап-проекту системи збирання та аналізування текстових даних з соціальних мереж.

В зв’язку з тим, що прямих аналогів системи не виявлено, то доцільно розвивати даний програмний комплекс як самостійний продукт для його монетизації та одержання доходів. Аналізування стартап проекту доводить успішність майбутнього розгортання проекту, а також надає початкові кроки для початку діяльності.

Визначена маркетингова стратегія стартапу дозволяє залучити широке коло інвесторі, що являється значним внеском для успішного запуску проекту. Така стратегія дозволить у короткі сроки визначити та залучити необхідну аудиторію юзери для майбутнього розвитку проекту.

ВИСНОВОК

У належності до завдання на магістерську дисертацію розроблено систему збирання та аналізування текстових даних отриманих з соціальних мереж, яка дозволяє збирати дані про актуальні інциденти у моменти їхнього утворення та постійного збору інформації про їх поточний стан. В результаті чого розроблено робочий прототип системи зі зручним юзер інтерфейсом.

В рамках магістерської дисертації проаналізовано існуючі рішення та роботи інших розробників систем аналізування текстової інформації та даних зібраних з соціальних мереж. Аналізування продемонструвало, що впровадження, розроблення та дизайн схожих програмних комплексів вимагає залучення якісного технічного забезпечення, проведення аналізування доменної області, залучення професійних розробників та інвестицій для імплементації системи якісного збирання та аналізування даних.

Розгорнуто та імплементовано робочий прототип програмного продукту на клауд хостингу. Обґрунтовано та вибрано та технічні інструментарії імплементації. Вибрано інструменти аналізування природних мов та супутніх технічних засобів.

В процесі перевірки системи отримано та обґрунтовано аналізування природних явищ та спортивних подій у вигляді діагрм, рендеринг яких виконує отримана система.

В систему закладено можливість масштабування. Виконано тестування системи на різноманітних рівнях, що підтверджує точність та правильність отриманих результатів, а також надійність системи і її відмовостійкість при навантаженнях. Автоматизовано процес розгортання системи, а також впроваджено та налаштовано безперервний інтеграційний процес.

Технічне завдання цілком задовольняється результатами роботи системи. Система виконує поставлене завдання збору та аналізування фактів чи подій запитуваних юзерами.

ПЕРЕЛІК ПОСИЛАНЬ

1. E. Adar Interacting with the ephemeral web. / E. Adar, M. Dontcheva, J. Fogarty, and D. S. Weld. Zoetrope // In UIST ’08 ACM Press, 2008. – P. 17-33.
2. P. Andre Continuum: designing timelines for hierarchies, relationships and scale. /P . Andre, M. L. Wilson, A. Russell, D. A. Smith, A. Owens. // In UIST ’07. ACM Press, 2007. – P. 24 – 25.
3. N. Bansal A system for online analysis of high volume text streams. / N. Bansal and N. Koudas. Blogscope. // In VLDB ’07. VLDB Endowment, 2007. P. 45 – 49.
4. Interactive Topic-based Browsing of Social Status Streams. / M. S. Bernstein, B. Suh, L. Hong, J. Chen, S. Kairam, and E. H. Chi. Eddi. // In UIST ’10, 2010. P. 15 – 30.
5. N. Diakopoulos Diamonds in the rough: Social media visual analytics for journalistic inquiry. /N. Diakopoulos, M. Naaman, and F. Kivran-Swaine. // In VAST 2010. P. 91 – 94.
6. Characterizing debate performance via aggregated twitter sentiment. /N. Diakopoulos and D. A. Shamma. // In CHI 2010. P. 102 – 110.
7. Visualization of linear time-oriented data: A survey. Web Information Systems Engineering. / S. F. Silva and T. Catarci. // In UIST ’10, 2010. P. 21 – 22.
8. S. Vieweg Microblogging during two natural hazards events: what twitter may contribute to situational awareness. / S. Vieweg, A. L. Hughes, K. Starbird, and L. Palen. // In CHI ’10. ACM Press, 2010. P. 24 27.
9. A. Java Why we twitter: Understanding the microblogging effect in user intentions and communities. / A. Java, X. Song, T. Finin, and B. Tseng. // In WebKDD, 2007. P. 11 – 19.
10. Adam Marcus TwitInfo: Aggregating and Visualizing Microblogs for Event Exploration / Adam Marcus, Michael S. Bernstein, Osama Badar, David R. Karger, Samuel Madden, Robert C. Miller // MIT CSAIL, 2011.
11. JDK 10 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/10/.

12. Maven Overview [Електронний ресурс] – Режим доступу до ресурсу: https://maven.apache.org/users/index.html.
13. Learn Git [Електронний ресурс] – Режим доступу до ресурсу: https://git-scm.com/.
14. Что такое travis-ci.org и с чем его едят [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/post/140344/.
15. Spring Boot Framework [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/.
16. Kafka Streams [Електронний ресурс] – Режим доступу до ресурсу: https://kafka.apache.org/documentation/streams/.
17. Kafka 1.1 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://kafka.apache.org/documentation/#api.
18. Create a Docker-based Service on DigitalOcean with Docker Compose [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@jmarhee/create-a-docker-based-service-on-digitalocean-with-docker-compose-44fa6e8f31a9.
19. Стартап [Електронний ресурс] // Wikipedia. – 2019. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%80%D1%82%D0%B0%D0%BF.
20. Техніко-економічні показники [Електронний ресурс] // Wikipedia. – 2019. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D1%96%D0%BA%D0%BE-%D0%B5%D0%BA%D0%BE%D0%BD%D0%BE%D0%BC%D1%96%D1%87%D0%BD%D1%96_%D0%BF%D0%BE%D0%BA%D0%B0%D0%B7%D0%BD%D0%B8%D0%BA%D0%B8.
21. Аудит [Електронний ресурс] // Wikipedia. – 2019. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%90%D1%83%D0%B4%D0%B8%D1%82.
22. Ступеневий аналіз конкуренції [Електронний ресурс] // Wikipedia. – 2018. – Режим доступу до ресурсу: https://studfile.net/preview/5992853/page:11/.

23. Модель п'яти сил конкуренції М. Портера [Електронний ресурс] // pidruchniki. – 2019. – Режим доступу до ресурсу: https://pidruchniki.com/12980108/marketing/analiz_konkurentiv.
24. Базова стратегія розвитку [Електронний ресурс] // studopedia. – 2019. – Режим доступу до ресурсу: https://studopedia.su/20_51020_bazovi-strategii-rozvitku-pidpriiemstv.html.
25. Класифікація споживачів [Електронний ресурс] // buklib. – 2019. – Режим доступу до ресурсу: https://buklib.net/books/22870/.
26. Примак Т. О. Стратегії позиціонування у теорії маркетингу [Електронний ресурс] / Тетяна Олександрівна Примак // http://mmi.fem.sumdu.edu.ua/. – 2012. – Режим доступу до ресурсу: http://mmi.fem.sumdu.edu.ua/sites/default/files/mmi2012_1_13_20.pdf.
27. Swot Analysis [Електронний ресурс] // wordstream. – 2019. – Режим доступу до ресурсу: https://www.wordstream.com/blog/ws/2017/12/20/swot-analysis.
28. Основні моделі ринку за типами конкуренції [Електронний ресурс] // studfile. – 2018. – Режим доступу до ресурсу: https://studfile.net/preview/7262847/.